Escuela Politécnica Superior

Aplicación Android para la búsqueda de datos abiertos mediante el uso de mapas

MapVieWide

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Alejandro Alemañ Arnau

Tutor/es:
Antonio Javier Gallego Sánchez

Julio 2020

Universitat d'Alacant
Universidad de Alicante

MapVieWide: Aplicación Android para la búsqueda de datos abiertos mediante el uso de mapas

Android Studio (Cliente) + API RESTful (Servidor)

Autor
Alejandro Alemañ Arnau

Director
Antonio Javier Gallego Sánchez
LENGUAJES Y SISTEMAS INFORMÁTICOS

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

ALICANTE, 30 de julio de 2020

Justificación y Objetivos

Hay veces que nos interesa saber información respecto a la ciudad donde nos encontramos, ya sea bien porque estemos de viaje o simplemente porque vivamos ahí y queramos saber más datos sobre el lugar. Con este propósito, he desarrollado una aplicación móvil para dispositivos Android a través de la cual los usuarios podrán obtener información estadística sobre la zona (densidad, envejecimiento, nacimientos, defunciones...) o bien sobre servicios (precio medio de los hoteles de una zona, precio del turismo rural...) mediante el uso de mapas. Por el momento únicamente está implementado para el país de España, pero se plantea desarrollar también para el resto del mundo.

Otro de los objetivos principales del proyecto es aprender las tecnologías en uso en el sector, como son Android, Lumen o Laravel. Para esto se va a implementar una arquitectura tipo cliente-servidor, mediante una aplicación Android que actuará como cliente y una API RESTfult implementada en Lumen para la parte de servidor. Además, también se pretende hacer uso de una metodología de desarrollo ágil la cual nos permita adaptarnos a los cambios y posibles problemas que nos vayan surgiendo.

Índice general

Índice de figuras

Índice de Listados

1. Introducción

En la actualidad estamos rodeados de información de todo tipo, la cual empleamos para reducir la incertidumbre o acrecentar el contenido que se tiene en una determinada área, contexto o situación, de ahí, que sea considerada como una herramienta para alcanzar conocimiento.

Mediante la aplicación Android desarrollada para este TFG (MapVieWide) obtendremos información que nos puede ser de utilidad, como podrían ser datos estadísticos sobre la población de una zona concreta (densidad, envejecimiento, nacimientos, defunciones, migración, sexo, etc.) sobre servicios (hoteles, zonas rurales, apartamentos, etc.) o incluso estadísticas territoriales (precio m2, total viviendas, viviendas vacías, etc.) así como otras estadísticas (contaminación, zonas verdes, etc.).

MapVieWide nos ayuda a la hora de recuperar información de nuestro interés de forma rápida, visualmente sencilla e intuitiva. Mediante los distintos gráficos que se muestran en la aplicación tendremos la posibilidad no solo de obtener la información que nos interese, sino también de comparar dichos datos con resultados nacionales o mostrar únicamente datos nacionales.

Como cliente tendremos una aplicación para el sistema operativo Android en su versión 7.0 o superior. Decidí realizar la aplicación en Android por afrontar el desarrollo de un proyecto con una nueva tecnología que nunca había tocado en la carrera, pero que siempre me había llamado la atención. Además tiene una gran comunidad de desarrolladores detrás, la documentación, las guías y las librerías son muy amplias y de fácil acceso para todo el mundo, incluso las que no son oficiales, sino creadas por el resto de la comunidad.

Por otra parte, Android cuenta con una gran tasa del mercado (en españa) que ronda el 85,1 % de dispositivos móviles mientras que iOS sólo representa el 14,9 %.

Como parte del servidor, tendremos una API RESTful desarrollada usando el framework de PHP Lumen en su versión 6.3.3. Elegí este framework de PHP ya que tras haber aprendido Laravel en la carrera, se me iba a hacer más sencillo aplicar Lumen, dado que es una versión reducida y más rápida de este.

Una parte importante dentro del desarrollo de la aplicación han sido las metodologías de desarrollo, dado que nos han ayudado a agilizar la creación de software, como explicaremos en apartados posteriores. En nuestro caso se ha utilizado la metodología Kanban, ya que no se requiere pertenecer a un equipo real para poder ponerla en práctica.

1.1. Objetivos

En este proyecto afronto un gran reto como es el desarrollo de una aplicación nativa para dispositivos Android. Partiendo de no tener ningún conocimiento sobre desarrollo para Android, será necesario realizar una fase de aprendizaje inicial, pasando por la fase de análisis de requisitos de la aplicación hasta llegar al desarrollo de una aplicación funcional.

El objetivo global de este proyecto es crear una aplicación Android como cliente, que utilice una API RESTful implementada en Lumen como servidor para la obtención de los datos a mostrar.

Un objetivo importante es la realización de la aplicación teniendo presente el control de las versiones de la misma utilizando Git, gracias al cual podremos realizar un desarrollo que se adapte a los cambios de manera rápida y tendremos la posibilidad de retomar en cualquier momento otra versión del proyecto.

Por otra parte, indicar que el desarrollo de aplicaciones móviles siempre ha sido algo que he deseado realizar durante el grado, pero que no he tenido oportunidad dado que es algo más especializado.

1.1.1. Desarrollo de software

El desarrollo de software es una parte muy importante dentro de un proyecto, ya que, puede que tengamos una buena toma de requisitos, unos objetivos claros y concisos sobre cómo se debe desarrollar la aplicación, pero si detrás de todo no hay un software robusto y eficiente, no conseguiremos los resultados deseados. Por lo tanto, con respecto al desarrollo de software, los objetivos específicos son los siguientes:

- Desarrollar un código limpio, modular y reutilizable.
- Mantener el código documentado con comentarios.
- Realizar todas las pruebas que sean posibles, no de forma exhaustiva pero sí un

mínimo de pruebas que nos garantice seguridad en el desarrollo.

- Hacer uso de las últimas versiones de librerías y frameworks.
- Gestionar el desarrollo mediante Git.

1.1.2. Funcionalidad de la aplicación

Con respecto a la funcionalidad de la aplicación, queremos conseguir una aplicación fluida, sencilla a nivel visual y usable para los usuarios finales, los objetivos específicos son:

- Desarrollar una aplicación que muestre, de forma clara, las fuentes de información disponibles en el mapa.
- Permitir al usuario acceder a la información y mostrar únicamente los datos que le interesan.
- Dar la posibilidad al usuario de comparar los datos mostrados con respecto a los distintos años y a su vez, entre los resultados nacionales y de las distintas comunidades.
- Mostrar los datos de una forma simple y concisa.

1.1.3. Metodología de desarrollo

Con respecto a la metodología de desarrollo de la aplicación, trataremos de utilizar el máximo número de herramientas ofrecidas y a su vez tratar de sacarle el máximo provecho a la metodología de desarrollo ágil utilizada. Los objetivos específicos son:

- Reunir todos los requisitos y tareas necesarias para el completo desarrollo de la aplicación.
- Utilizar una herramienta de control de versiones, como podría ser Git e intentar utilizar todas las herramientas que nos ofrece (Issues, Hitos, Wiki...)
- Ser capaz de ver el estado general de la aplicación de un vistazo (mediante los tableros, hitos...)

- Versionar la aplicación y lanzar varias releases de la misma según sea necesario (añadir nuevas features, arreglar bugs...)

2. Estado del arte

En esta sección vamos a analizar algunas aplicaciones similares que podemos encontrar en el mercado, incluyendo tanto aplicaciones para dispositivos móviles como para web.

Posteriormente veremos cómo deberíamos enfocar la aplicación para cubrir las funcionalidades de estas aplicaciones que gustan a los usuarios y ver qué podríamos mejorar y en qué nos diferenciaríamos.

También hay que tener en cuenta que, a la hora de realizar la propuesta del trabajo, intenté que fuese una aplicación nueva, que no tuviese competencia directa, por ello, buscar aplicaciones similares será algo complicado, pero sí podemos encontrar unas cuantas que se podrían asemejar a la aplicación que estamos desarrollando.

2.1. Portal Esri

Portal Esri[1] es un portal web de datos abiertos que contiene información a nivel nacional mediante el cual se puede sacar información de todo tipo, bien sea para consultar, analizar, descargar, generar otras aplicaciones o servicios con dichos datos.

Portal Esri no solamente ofrece datos tratados por Esri España, también es una recopilación de los diferentes portales Open Data existentes en la actualidad.

Los datos se pueden mostrar mediante el uso de mapas o bien navegando entre las distintas listas de datos que contiene el portal.

Las distintas categorías de datos que ofrece este portal son los siguientes:

- Comercio, empresas y negocios.
- Datos sociodemográficos.

[1] `http://opendata.esri.es/`

- Economía.
- Educación, cultura, turismo y ocio.
- Gobierno, justicia y defensa.
- Salud.
- Ciencia, tecnología e industria.
- Territorio y medio ambiente.
- Transporte
- Urbanismo e infraestructuras.

Figura 2.1.: Portada del Portal Esri.

2.2. OsmAnd

OsmAnd es una aplicación móvil para Android e iOS que trabaja con un mapa y una aplicación de navegación con acceso a datos de OpenStreetMap (OSM), el cual es un proyecto colaborativo para crear mapas editables y libres, fundado en julio de 2004 y totalmente funcional hasta día de hoy.

OsmAnd ofrece diversas visualizaciones en el mapa como podrían ser:

- Mostrar POI (puntos de interés) a tu alrededor.
- Mostrar mapas especializados de teselas.
- Mostrar superposiciones diferentes como trazas de navegación GPX y mapas adicionales.
- Puntos de interés de Wikipedia.
- Imágenes a nivel de calle.

Figura 2.2.: Icono de la aplicación de OsmAnd.

2.3. Waze

Waze es una aplicación móvil que trabaja con mapas, es social y retroactiva, lo que quiere decir que algunos de los datos que muestra la aplicación son reportados por usuarios que conocen esa información y la publican.

Waze ofrece diversas visualizaciones en el mapa como podrían ser:

- Avisos de tráfico, policía, peligros y mucho más en tu ruta.
- Cambios de ruta instantáneos para evitar tráfico y ahorrarte tiempo.
- Los datos mostrados se basan en datos de tráfico real.
- Encontrar la gasolinera más barata en tu ruta.

Figura 2.3.: Icono de la aplicación Waze.

2.4. Conclusiones

Como hemos podido observar, ya existen varias aplicaciones que ofrecen la posibilidad de obtener datos abiertos sobre diversas categorías:

- Portal Esri: es un portal que contiene mucha información sobre el mapa, sin embargo, obtener los datos no es algo tan sencillo para un usuario que no abarque los suficientes conocimientos.

- OsmAnd: es una aplicación móvil que nos ofrece diversas visualizaciones en el mapa, como puntos de interés o mapas especializados, pero no logra el objetivo que buscamos, como podrían ser, datos estadísticos sobre las distintas comunidades, poblaciones, etc. (natalidad, densidad, mortalidad, precios hoteleros...).

- Waze: es una aplicación móvil que nos ofrece diversas utilidades sobre el mapa, como podrían ser datos sobre tráfico, gasolineras con precios asequibles, etc. La aplicación que estamos desarrollando sigue buscando un enfoque distinto al que nos ofrece esta aplicación.

Tras ver los servicios que nos ofrecen las distintas aplicaciones existentes en el mercado, la aplicación que vamos a desarrollar estará orientada a la obtención de datos (estadísticos, demográficos...) relativos a las distintas comunidades autónomas de España, tratando de mostrar dichos datos de manera ordenada y sencilla para el usuario final.

3. Metodología

Hacer un desarrollo de calidad no aporta valor directo al producto final, pero si de forma indirecta. Un proceso de desarrollo de calidad aporta reducción en el tiempo de respuesta, cuando hay cambios en las especificaciones o aparecen errores en el sistema, y por lo tanto reducción en los costes del desarrollo.

Para empezar, la metodología a emplear debe de ajustarse a las características del proyecto (especificaciones, tiempo de vida, equipo de desarrollo...). Usar una metodología de desarrollo en cascada no es conveniente ya que no se ajusta a la realidad del proceso de desarrollo de software y, cuando se tiene poca práctica, al hacer un análisis completo de la aplicación es muy fácil cometer errores que después implican retrasos en el desarrollo. Por lo tanto se ha decidido utilizar una metodología ágil.

Dentro de las principales metodologías ágiles encontramos Scrum y Kanban. La metodología Scrum no tendría mucho sentido en un proyecto realizado por una sola persona, ya que en Scrum:

- Existen roles para los miembros del equipo.
- Se realizan daily meetings, reuniones diarias en las que se comenta el trabajo realizado el último día y los distintos problemas encontrados en el desarrollo realizado por el equipo.

Por lo tanto la metodología elegida ha sido Kanban, la cual nos ofrece una mayor flexibilidad. Mediante el sistema de issues de GitHub he podido establecer las tareas a realizar y gracias a la opción de Proyectos he podido establecer un tablero Kanban en el cual tener una visión general del estado del mismo.

Mediante Kanban hacemos uso de tableros para gestionar, de manera visual, la realización de determinados procesos o tareas. A parte, Kanban es una metodología sumamente fácil de implementar en un proyecto y muy eficaz a la hora de repartir el trabajo restante, ya que nos permite observar de manera clara qué se está haciendo ahora, qué se ha terminado y qué se debe hacer a continuación.

Dicho tablero contiene tantas columnas como estados en los que pueda encontrarse un

proyecto. Normalmente se tienen 3 estados: *To Do, In progress, Done.*

En Kanban existen una serie de fundamentos que deben seguirse para que la metodología se establezca de la forma correcta:

- Visualizar el flujo de trabajo: El tablero ayuda a visualizar todas las tareas del proyecto, tanto realizadas como por realizar. Esto permite al equipo tener una visión más clara del estado del desarrollo.

- Limitar la cantidad de Trabajo en Proceso: Establecer metas asequibles es de gran importancia. Un exceso de trabajo puede acarrear desmotivación entre los miembros del equipo y conllevar a no poder terminarlas.

- Lectura fácil de indicadores visuales: El uso de tarjetas de colores facilitará enormemente la distinción entre tipos de trabajo.

- Realizar un seguimiento del tiempo: Evaluar de forma continuada si se está cumpliendo con los plazos de desarrollo y entrega, y ayudará a comprender mejor el estado del proyecto.

A modo de resumen, utilizar una metodología ágil nos ayuda a dimensionar mejor los proyectos minimizando los riesgos, divide las responsabilidades dentro del equipo, aumenta la autonomía y la transparencia en el mismo, también incrementa el valor añadido y la predictibilidad de los resultados, por último, nos ofrece una respuesta rápida a los cambios.

3.1. Sistema de control de versiones

Para el sistema de control de versiones he utilizado Git[1]. En este caso se han utilizado dos ramas, la master y la de hotfix para arreglar los posibles bugs que van surgiendo. No he considerado hacer más ramas para el desarrollo dado que no iba a ser un proyecto muy grande.

- master: rama con el código que estaría en producción. Código estable y validado.

- hotfix: rama con el código de un commit de master la cual se bifurca para arreglar el error y mergearse más tarde de nuevo con la rama master para solucionar el bug.

[1]Web oficial de Git. https://git-scm.com/

Normalmente, a la hora de comenzar una nueva tarea o una serie de tareas que tienen bastante en común, se crea una rama para el desarrollo de dichas features. En nuestro caso, tenemos únicamente la rama master y la de hotfix, por lo tanto conforme vayamos avanzando en el desarrollo de la aplicación por la rama master, cualquier bug o problema que nos encontremos, crearemos una rama de hotfix para solucionar dicho error y más tarde volver a mergearlo con la rama principal.

En la imagen 3.1 podemos ver como se trabaja con una rama hotfix, en la cual primero se coge un commit de la rama master y más tarde se mergea de nuevo con master en un nuevo commit con el bug solucionado.

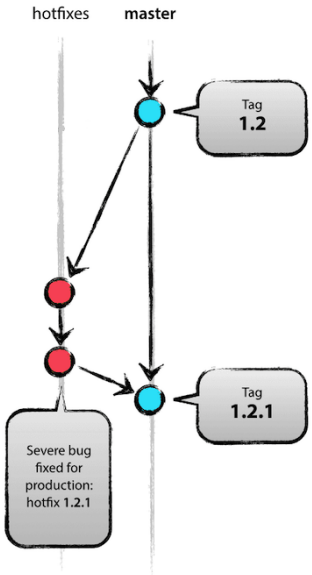


Figura 3.1.: Merge de una rama hotfix con master.

3.2. Issues

El issue es la forma en la que se pueden crear las tareas e incidencias en GitHub y gracias a ellos podemos llevar un seguimiento del trabajo pendiente.

Se pueden crear etiquetas con las que categorizar los issues para entender rápidamente de qué se trata. En mi caso he creado tres etiquetas adicionales para diferenciar los issues:

- enhancement: para tareas opcionales, son tareas que no influyen en el funcionamiento de la aplicación. No aportan mucho valor al producto final, normalmente incluyen una funcionalidad a futuro (diseño, mejoras visuales...).

- features: tareas que añaden funcionalidad y/o valor al producto final.

- fix: tareas para solucionar bugs del desarrollo. Los issues con esta etiqueta explican el problema y una posible solución.

- release: etiqueta para los merges (pull request) de hotfix sobre master. Para la solución en producción de los bugs encontrados.

Al crear el issue, le añadiremos los siguientes elementos:

1. Título descriptivo de la tarea a realizar.

2. Descripción de la tarea con un resumen de lo que se debe hacer. Si es un bug, se debería de indicar como replicar el fallo.

3. Persona asignada a resolver el issue. Si fuese un equipo de trabajo, se asignaría al responsable que repartiría las tareas.

4. Etiquetas que categorizan el issue (enhancement, features, fix, release...). Por lo general tenemos varios tipos de etiquetas para diferenciarlos a la hora de buscar algún tipo en concreto. (Ver imagen 3.4).

5. Proyecto: indica el tablero donde se van a insertar los issues (ver sección 4.1.3.4).

6. Hito al que pertenece (ver sección 3.3). He usado el nombre de la versión en la que va a ser publicada para poder saber los issues que van a ser resueltos en esa versión.

En la figura 3.2 se puede ver un ejemplo de un issue en el cual se indica el título descriptivo de la tarea a realizar. Debajo encontramos una descripción de la tarea a realizar. Si fuera un fallo encontrado, se deberían de indicar los pasos para poder reproducir el error. En el lado derecho podemos observar el responsable de la tarea, la etiqueta que le ha sido puesta, el proyecto al que ha sido asignado y el hito al que pertenece.

Debajo de la descripción se irían añadiendo mensajes con los cambios que va sufriendo el issue, como por ejemplo, un cambio de responsable o el hito al que pertenece. Junto a estos mensajes, se pueden ir añadiendo comentarios que podrían ser necesarios en un futuro.

Figura 3.2.: Ejemplo issue.

En la imagen 3.3 podemos observar cómo se realiza un filtrado en GitHub. En la parte superior encontramos el filtro aplicado actualmente, el cual está filtrando los elementos que son issues (también podrían ser pull request), que están abiertos y contienen la etiqueta "features". Al comienzo del listado de issues, como podemos ver en la imagen, también se nos permite filtrar por autor, etiquetas, proyectos...

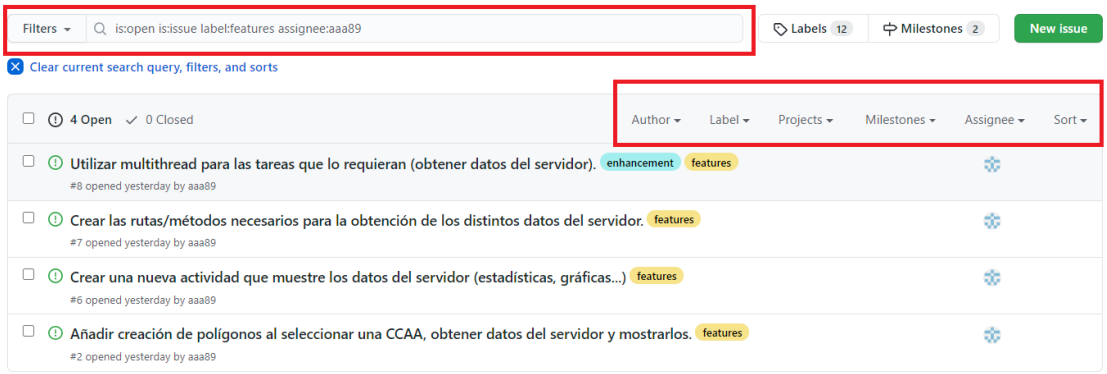


Figura 3.3.: Ejemplos de filtros y ordenación de los issues en GitHub.

Durante el proceso de gestión del proyecto, voy creando issues y asignándolos a los distintos hitos que he creado.

Estos hitos tienen una fecha marcada desde el inicio para ser completados (fecha de vencimiento). En el caso de ver que no nos da tiempo, siempre podemos eliminar alguna tarea o cambiarla por alguna que nos consuma menos tiempo para realizarla y de esta forma no retrasar la entrega inicial del hito.

En nuestro caso, como solamente utilizamos la rama master para todo el desarrollo, no debemos de crear una rama con la nueva feature y cerrarla más tarde mediante un pull request con la rama master.

Lo que sí se ha hecho ha sido comentar en los distintos commits de la rama master los errores encontrados, que se tendrán que solucionar más tarde, y de los cuales se ha creado un issue con la etiqueta "fix"para tenerlos presentes. En este caso lo mejor es que en cuanto se encuentra un error, escribamos toda la información que tenemos sobre él, ya sea bien las condiciones necesarias para poder reproducirlo como una posible solución al problema.

Para llevar un buen seguimiento de qué cambios se han hecho, se deben de relacionar los distintos issues con los commits. Para ello, a la hora de hacer el commit, añadimos al mensaje de este una de las palabras clave de GitHub: closed, fix, resolved... y el número del Issue al que queramos referenciarlo. Por ejemplo: Resolved #7, para indicar que dicho commit soluciona el issue número 7.

Si a la hora de hacer el commit se nos ha olvidado añadir qué issue cierra, podremos añadir un comentario al issue con la URL al commit. Por ejemplo: Close URL (hacia el commit que trata).

En la imagen 3.4 podemos ver un ejemplo de un comentario en un issue que cierra un commit en el que no se comentó a qué issue hacía referencia, en este caso lo referenciamos en un comentario dentro del issue para tener claro qué commits tienen relación con el issue. Añadimos 'Closes' y a continuación el commit o los commits que está cerrando, en este caso *Closes #3, #4, #5*

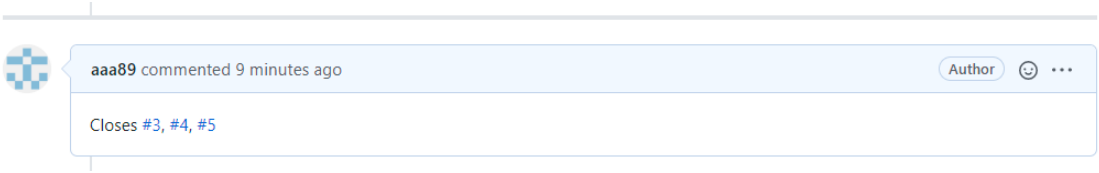


Figura 3.4.: Ejemplo de issue cerrado referenciando los commits que cierra.

3.3. Hitos

Los hitos (milestones en GitHub) nos ayudan a establecer puntos de control en los que podemos agrupar los issues, indicando qué issues tienen que estar cerrados cuando lleguemos a la fecha del hito (figura 3.5).

Se ha utilizado un repositorio distinto para cada parte del proyecto, un repositorio para el cliente y otro para el servidor.

En este proyecto he utilizado los issues para establecer las versiones a entregar y en qué fechas aproximadas se realizarían dichas entregas. He realizado de momento 3 hitos para el cliente y 3 hitos en paralelo para el servidor. Los cuales se han ido desarrollando según avanzaba la aplicación, con las distintas versiones que he ido generando; una inicial, otra con la mayor cantidad de features realizadas y otras características opcionales añadidas, y por último una para correcciones de código y añadir algunas features opcionales.

Esta metodología ágil me ha permitido realizar múltiples entregas consiguiendo así un flujo constante de trabajo, de entregas y de progreso en el desarrollo.

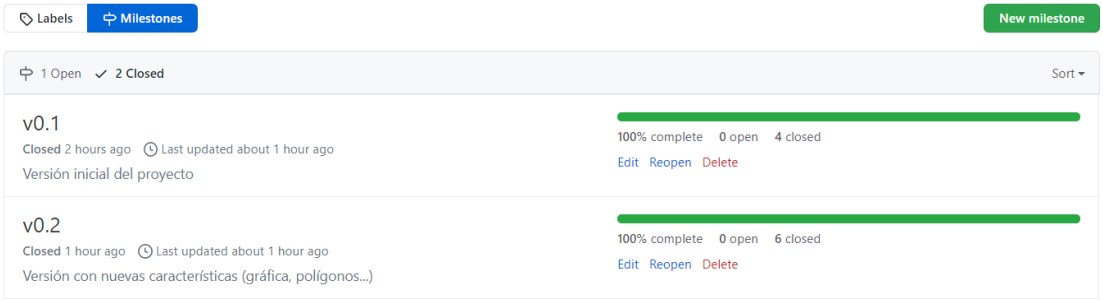


Figura 3.5.: Ejemplo de hitos creados ya cerrados.

3.4. Versionado

Dado el tiempo que he tenido, he decidido repartir el proyecto en estas versiones, tanto del cliente como del servidor:

- v0.1 (01/05/2020): En esta primera versión para el cliente creamos las áreas que se iban a tratar en el mapa y se recuperaron los datos del servidor, sin mostrarlos todavía. Para el servidor en esta primera versión se trataron de añadir los puntos geométricos que necesitaría el cliente.

- v0.2 (20/07/2020): En esta segunda versión añadimos una nueva actividad que muestra los datos recogidos del servidor en una gráfica y tenemos la opción de seleccionar los datos que deseemos. En este punto ya tenemos una aplicación funcional mediante la cual poder recuperar los datos que nos ofrece el servidor. Para el servidor en esta segunda versión ya se recuperaron los datos sobre las distintas comunidades autónomas.

- v.03 (08/09/2020): En esta última versión se corregirán los bugs encontrados a lo largo del desarrollo y se incorporarán las últimas características opcionales que consideremos, además de mejorar el aspecto visual de la aplicación. En esta tercera versión se optimizaron algunas peticiones hacia el servidor por ejemplo en la cantidad de puntos que se traían de los puntos geométricos o eliminar algunos datos vacíos que no aportaban información.

3.5. Diseño del cliente

Para realizar el primer diseño de la aplicación he usado el programa Balsamiq (ver sección 4.1.4). Estos diseños me permitieron crear un primer boceto de la interfaz de la aplicación así como analizar los requisitos de la misma.

Figura 3.6.: Mockup de inicio aplicación.

Cuando el usuario entra a la aplicación por primera vez se encuentra directamente el

mapa con ciertos puntos de interés, en este caso se mostrarían las distintas comunidades autónomas de las que se puede obtener información (figura 3.6).

También, como parte opcional, tenemos distintos botones alrededor del mapa que nos sirven, por ejemplo, para hacer zoom, activar modo noche, geolocalización, etc.

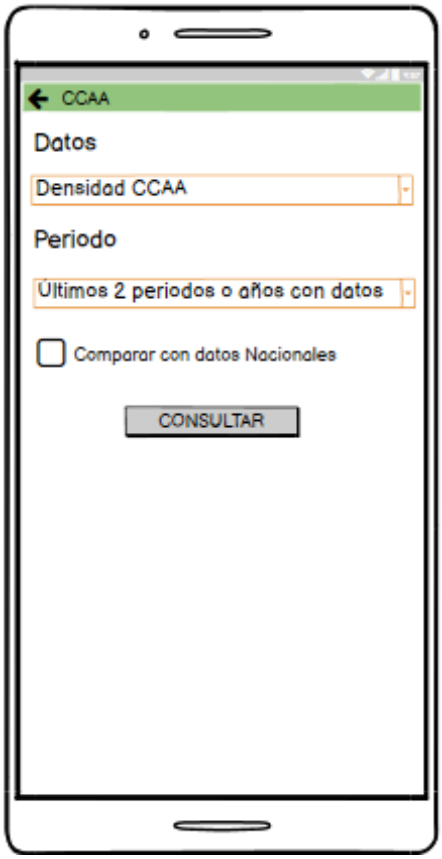


Figura 3.7.: Mockup sobre la actividad para obtener los datos.

Como podemos observar en la figura 3.7, tenemos una pantalla donde podemos obtener distintos datos de la aplicación, así como obtener otros datos, compararlos con los datos nacionales, modificar el período de los datos, etc.

Figura 3.8.: Mockup sobre la actividad para obtener los datos.

Al clicar en el botón de consultar, obtendremos los datos que buscábamos en forma de gráfica, ya sea bien por comunidad autónoma, o comparando estos datos obtenidos de la CCAA con los datos nacionales de ese período seleccionado.

3.6. Diseño del servidor

A continuación vamos a describir como está compuesto el servidor de forma breve, ya que más adelante se explicará de forma detallada.

En este caso no tenemos diversos usuarios en la aplicación, únicamente existe un administrador que podrá acceder para gestionar y monitorizar el servidor. Todos los datos que se van a obtener en el API se guardarán en memoria mediante un sistema de cache y se usarán para enviársela directamente al cliente cuando este los solicite.

Los principales *endpoints* que tendrá el servidor para suplir la demanda del cliente serán los siguientes:

- Puntos geométricos obtenidos de OpenStreetMap[2] para dibujar las áreas en el mapa de las distintas comunidades autónomas.

- Datos obtenidos de la web INE[3] (Instituto nacional de estadística) que nos reportará todos los datos necesarios sobre población, economía y demás que necesitará mostrar el cliente Android.

3.7. Diseño de servicios

En este apartado hablaremos sobre las fuentes de información que contendrá el API y que se mostrarán en el cliente.

- Web del INE: De aquí se han sacado la mayor parte de los datos que son mostrados en la aplicación, ya que es una web muy completa que contiene todo tipo de información respecto a población, datos demográficos, economía, etc.

- Mapa de OpenStreetMap: De esta herramienta se han sacado todos los puntos geográficos que se muestran en el mapa, principalmente áreas de las distintas comunidades autónomas en un formato reducido de puntos para optimizar el rendimiento.

[2]Web de OpenStreetMap: `www.openstreetmap.org`
[3]Web del INE: `https://www.ine.es/`

4. Tecnologías

En esta sección vamos a presentar las tecnologías utilizadas para el desarrollo del proyecto. En primer lugar se describen las tecnologías utilizadas en el cliente y seguidamente veremos las empleadas en el desarrollo del servidor.

4.1. Cliente

La elección de las tecnologías en el cliente han requerido un poco más de trabajo de lo habitual, ya que era una tecnología que no había tocado nunca antes, pero desde un inicio sabía que tenía claro que quería trabajar con una aplicación Android y a parte de eso, elegí un tutor acorde a mis necesidades, que tuviese los conocimientos necesarios para guiarme y/o aconsejarme cuando no supiese como abarcar un tema.

Para la elección de las librerías dentro de la propia aplicación, hice un análisis previo sobre qué librerías iba a necesitar, bien para obtener datos del servidor, para desarrollar el propio proyecto, para manejar los mapas que iba a utilizar, etc. En mi caso, he utilizado las que se nombran a continuación, teniendo en cuenta también el IDE (Integrated Development Environment) con el que he trabajado y el lenguaje de programación utilizado.

4.1.1. Android y Java

Para el desarrollo del cliente se ha utilizado el lenguaje Java, en concreto, la versión de Java 8, la cual nos permite mejorar el rendimiento para ciertas operaciones y acotar algunos aspectos del código. La versión de Java ha sido utilizada para el sistema operativo Android en su versión 7.0 en adelante.

Android es el sistema operativo más utilizado a nivel mundial en la actualidad, cuenta con más de 2500 millones de dispositivos activos en todo el mundo.

He decidido utilizar la versión de Android 7.0 (Nougat) para este proyecto ya que

ofrece una alta compatibilidad con la mayoría de dispositivos (73,7 %) y nos permite utilizar algunas librerías en una versión superior, lo cual nos ayuda en el desarrollo.

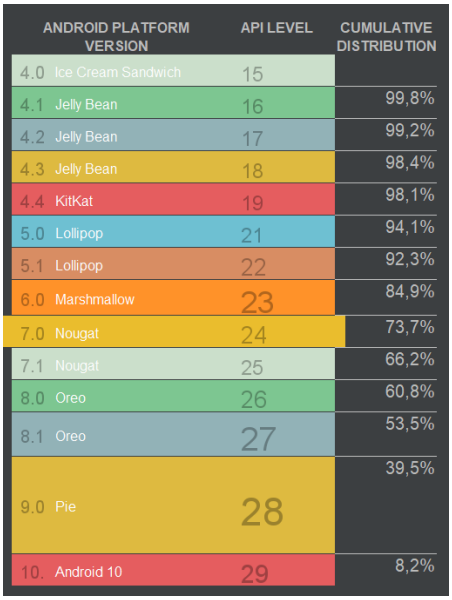


Figura 4.1.: Grado de compatibilidad entre versiones Android.

Además, también he decidido utilizar esta versión ya que el mayor porcentaje de los usuarios tienen versiones entre Android 5.1 (Lollipop) y 9 (Pie), como podemos observar en la imagen 4.2, casi un 90 %[1] de los usuarios se concentran en esas versiones.

[1]Datos de la web de Android. `https://developer.android.com/about/dashboards`

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

89,3%

Figura 4.2.: Nivel de distribución entre versiones Android.

Otro aspecto importante a tener en cuenta de Android, es que tiene una documentación[2] muy extensa para el desarrollo de una aplicación. Algunas de las cosas que nos enseña son las siguientes:

- Diversos apartados sobre rendimiento y seguridad.
- Buenas prácticas en testing.
- Guías de diseño para aumentar la usabilidad de la aplicación.

También cuenta con una larga lista de repositorios que contienen muchos códigos de ejemplo que nos facilita el aprendizaje, además, tiene muchas librerías[3] tanto propias como creadas por parte de la comunidad de usuarios que nos pueden ser de utilidad.

4.1.2. Android Studio

Para la implementación del código del cliente he decidido utilizar el IDE Android Studio en su versión 3.6.3, ya que es el entorno de desarrollo oficial de Google para aplicaciones de Android.

[2]Documentación para desarrolladores. `https://developer.android.com/docs`
[3]Cuenta de Google Samples en GitHub. `https://github.com/googlesamples`

Este IDE está basado en otro IDE, conocido como IntelliJ IDEA[4], el cual es uno de los entornos más utilizados y conocidos para el desarrollo Java. Dicho IDE tiene muchísimas herramientas que ayudan a la hora de escribir código, y además nos ofrece la posibilidad de personalizarlo como queramos.

Google va modificando el entorno añadiéndole más herramientas, por ejemplo, la vista de diseño de interfaz de usuario que se muestra en la aplicación (ver figura 4.3) o la ejecución de la aplicación sin necesidad de un dispositivo físico gracias a un emulador, el cual, podremos personalizar a nuestro gusto (versión de Android, tablet o móvil, etc.)

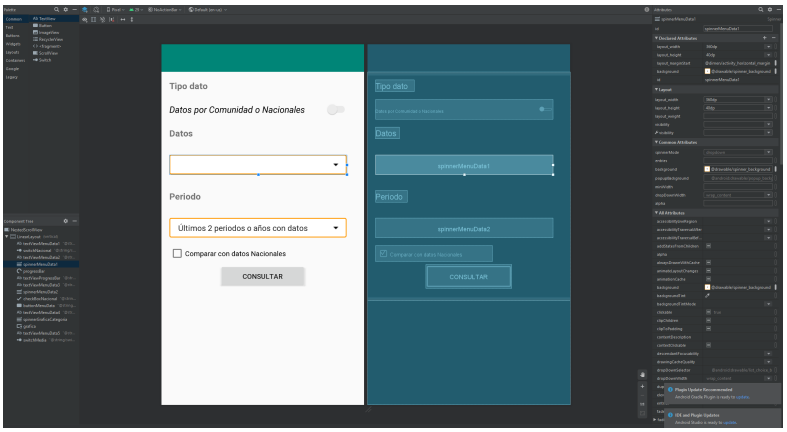


Figura 4.3.: Herramienta de edición de interfaz de usuario en Android Studio (Design + BluePrint).

4.1.3. Librerías y APIs

En este apartado comentaremos todas las librerías y APIs tanto oficiales como de la comunidad que hemos utilizado para el desarrollo de la aplicación.

4.1.3.1. APIs de Google Maps

A continuación se indicarán las APIs de Google Maps[5] que se han utilizado en el desarrollo del proyecto.

[4]ItelliJ IDEA `https://www.jetbrains.com/idea/`
[5]APIs de Google maps. `https://console.cloud.google.com/marketplace`

4.1.3.1.1. Places API Mediante Places API somos capaces de realizar las siguientes funciones dentro del mapa:

- Obtener datos de la base de datos utilizada por Google Maps.
- Autocompletado de lugares que se puede usar para completar automáticamente el nombre y/o la dirección de un lugar a medida que vamos escribiendo en el buscador.
- El autocompletado de consultas se puede utilizar para proporcionar un servicio de predicción de consultas para búsquedas geográficas basadas en texto y devolver consultas sugeridas conforme se escribe.

4.1.3.1.2. Maps SDK para Android Maps SDK nos permite agregar mapas basados en datos de Google Maps. El SDK maneja automáticamente el acceso a los servidores de Google Maps, la visualización del mapa y la respuesta a los gestos del usuario, como los clicks y los arrastres.

4.1.3.1.3. Geocoding API Esta API nos ayuda a convertir direcciones en coordenadas geográficas (geocodificación), que se puede utilizar para colocar marcadores o posicionarse en el mapa. Esta API también nos permite convertir coordenadas geográficas en una dirección (geocodificación inversa).

4.1.3.1.4. Geolocation API Mediante esta herramienta podemos encontrar nuestra ubicación en un radio de precisión basada en la información de torres celulares y en los puntos de acceso WiFi que un cliente móvil puede detectar. Este servicio se usa principalmente cuando el GPS no es posible o apropiado.

Figura 4.4.: Logo del conjunto de APIs de google.

4.1.3.2. Retrofit

Retrofit[6] es un cliente REST para Android y Java, desarrollada por Square, muy simple y fácil de aprender. Permite hacer peticiones GET, POST, PUT, PATCH, DELETE y HEAD, gestionar diferentes tipos de parámetros y parsear automáticamente la respuesta a un POJO (Plain Old Java Object).

En nuestro caso, para realizar la conexión entre el cliente y la API Rest, hemos hecho uso de dicha librería que convierte las peticiones realizadas a una API Rest en una interfaz de Java y nos devuelve un objeto o una colección de objetos, utiliza la librería Gson para la serialización y deserialización entre objetos Java y su representación en notación JSON.

Un ejemplo de la definición de la interfaz a un método para realizar llamadas a una API Rest podría ser así:

Listado 4.1: Ejemplo de una interfaz llamando a la API Rest mediante Retrofit.

```
1 public interface ApiDataService {
2     @GET("natalidadCCAA/{idCCAA}/{periodo}")
3      Call<MapVieWideResponseDatos> getNatalidadCCAA(@Path("idCCAA
          ") String idCCAA, @Path("periodo") String periodo);
4 }
```

La etiqueta o decorador que se pone encima del método (@GET en este caso) indica el tipo de petición HTTP a realizar. Como ya comentamos antes, según la documentación[7] de Retrofit, soporta los siguientes métodos: GET, POST, PUT, PATCH, DELETE y HEAD. Realizar una llamada al método (ver listado 4.1) generará una petición HTTP GET a la ruta "/natalidadCCAA/idCCAA/periodo/". En este caso tenemos dos parámetros que son variables en la ruta, tanto "idCCAA" como "periodo", las cuales se completarán o sustituirán al realizar la llamada al mismo.

A continuación se muestra un ejemplo de la respuesta JSON que es devuelta por la API para la petición anterior:

Listado 4.2: Ejemplo de la respuesta por parte de nuestra API.

```
1 {
2    "results":[
3       {
4          "categoria":"Fecundidad. Canarias.",
5          "datos":[
```

[6]Librería Retrofit. https://square.github.io/retrofit/

[7]Documentación Retrofit. `https://square.github.io/retrofit/#api-declaration`

```
            {
               "year":2019,
               "valor":6.36
            },
            {
               "year":2018,
               "valor":6.75
            }
         ]
      }
   ],
   "href":"\/natalidadCCAA\/9001\/2"
}
```


La respuesta en formato JSON sería mapeada automáticamente por Retrofit a una clase Java, esto debemos tenerlo en cuenta a la hora de programarlo, ya que debe contener exactamente la misma estructura sino no recuperará los datos del servidor correctamente. En nuestro caso, la clase Java tiene la siguiente estructura:

Listado 4.3: Ejemplo de la estructura de la clase Java.

```
public class DatosApi {
  private String categoria;
  private List<Dato> datos;
}
```

A su vez, tenemos la clase Java "Dato", que contiene lo siguiente:

Listado 4.4: Ejemplo de la estructura de la clase Dato de Java.

```
public class Dato {
  private String year;
  private Object valor;
}
```

En caso de que la petición y la respuesta sea correcta, el método nos devolverá una instancia de "Call<MapVieWideResponseDatos>".

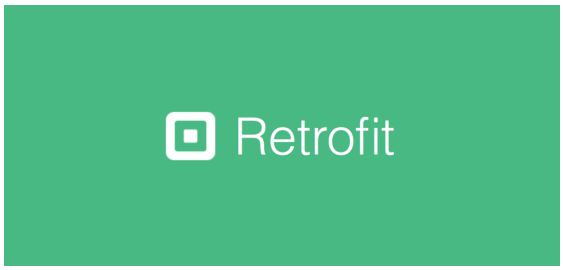


Figura 4.5.: Logo de Retrofit.

4.1.3.3. MPAndroidChart (Comunidad)

MPAndroidChart[8] es una librería de gráficos de Android, es compatible con gráficos de líneas, barras, radares, burbujas y velas, así como escala, panorámica y animaciones.

En nuestro caso la hemos utilizado para mostrar en el cliente los datos obtenidos del servidor en un gráfico de barras, un gráfico combinado que contiene tanto el gráfico de barras como un gráfico de líneas para otros parámetros.

Listado 4.5: Clases utilizadas de la librería MPAndroidChart.

```
1    private CombinedData graficaCombinada;
2    private LineData datosMedia;
3    private LineDataSet lineDataSetMedia;
4    private BarData datosBar;
5    private BarDataSet barDataSetCCAA;
6    private BarDataSet barDataSetNacional;
```

Necesitamos una clase *LineDataSet* y *BarDataSet*, la primera para guardar el conjunto de puntos que irán en la gráfica lineal y la segunda para los puntos o datos del gráfico de barras, después utilizamos las clases *LineData* y *BarData* para guardar dicho conjunto de puntos y por último lo inyectamos en una clase *CombinedData* que contiene todos los tipos de gráficos a mostrar, que en nuestro caso será únicamente el lineal y el de barras.

[8]Repositorio de MPAndroidChart. `https://github.com/PhilJay/MPAndroidChart`

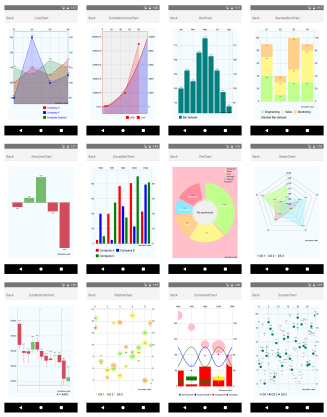


Figura 4.6.: Tipos de gráficos de MPAndroidChart.

Figura 4.7.: Logo de MPAndroidChart.

4.1.4. Balsamiq

A la hora de diseñar la interfaz de nuestra aplicación siempre surgen dudas, sobre cómo quedará, sobre cómo cuadrar los elementos, etc. Todos estos problemas pueden ser resueltos mediante alguna herramienta para crear wireframes, que son diseños básicos y minimalistas que nos pueden ser de ayuda a la hora de recrear lo que tenemos en mente o simplemente para darnos la idea que no encontramos sobre el diseño.

Después se pueden crear mockups para que el cliente pueda validar si lo que se ha pensado le gusta o habría que modificar cosas sobre el diseño. Estos diseños son de gran utilidad, ya que no cuestan mucho tiempo de realizarse, son fáciles de modificar y nos permiten plasmar un diseño completo de una aplicación, haciendo incluso un tour de las pantallas que tendría nuestra aplicación y con posibilidad de navegar entre las mismas como si estuviésemos en la propia aplicación, además nos ayuda a encontrar posibles problemas que suelen surgir en la fase de diseño y que podrían surgir más adelante en el desarrollo.

Para este proyecto he decidido utilizar la herramienta Balsamiq[9] ya que hemos trabajado con ella a lo largo de la carrera y la vi de gran utilidad.

[9]Web de Balsamiq. `https://balsamiq.com/wireframes/`

Los diseños de balsamiq [?] son de baja fidelidad, reproduce la experiencia de dibujar en un bloc de notas o pizarra, pero usando el ordenador, te obliga a concentrarte en la estructura y el contenido principalmente, evitando tener que elegir colores para la aplicación y detalles que deberían venir más adelante en el desarrollo.

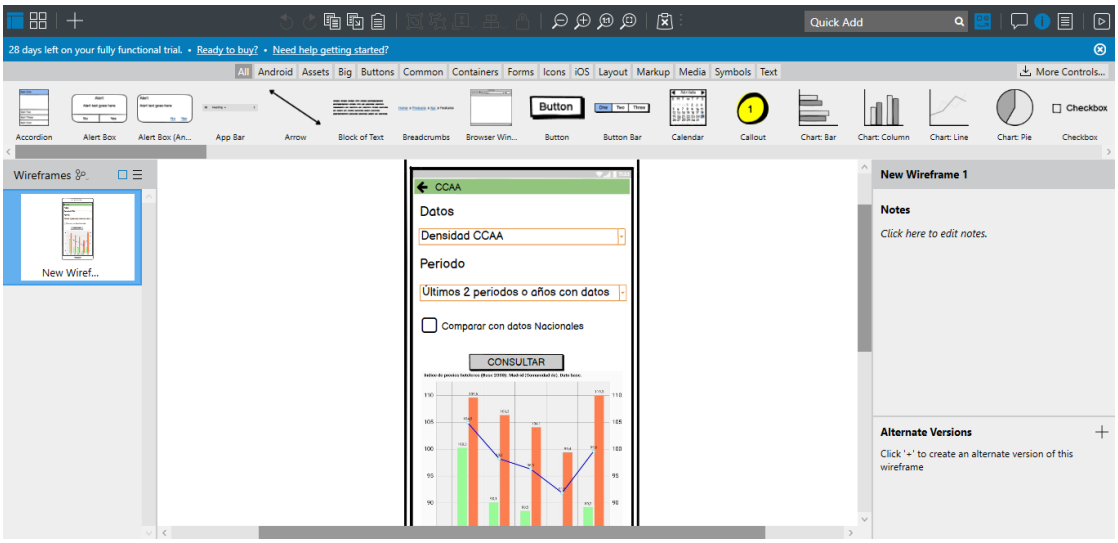


Figura 4.8.: Herramienta Balsamiq.

4.2. Servidor

Para la elección de la tecnología de servidor inicialmente evalué la posibilidad de utilizar alguna de las herramientas que había visto durante la carrera, como Laravel o Node.js. Finalmente, tras discutirlo con mi tutor, decidí utilizar Lumen, un framework PHP ligero basado en Laravel que es especialmente adecuado para el desarrollo de APIs Rest eficientes.

Por lo tanto, al haber tratado ya con Laravel en la carrera y ver las posibilidades que tenía como framework y, a parte, comentarme mi tutor que podría ser buena idea, me he decidido por hacer la API Rest con esta tecnología.

4.2.1. Lumen/PHP

Lumen[10] es un framework PHP con el cual podemos crear una API Restful de forma sencilla y rápida. Lumen, al ser una versión reducida, es muy rápido. Además nos ofrece la posibilidad de añadir funcionalidades de Laravel en el caso de ser necesario.

En nuestro caso hemos decidido no utilizar base de datos, sino llamadas cacheadas. A la hora de cachear las llamadas, tanto Lumen como Laravel, ofrecen un cacheado interno a las distintas URLs de las rutas que tenemos en el servidor. Por lo tanto no hemos tenido que tratar con ningún plugin adicional como podría ser Eloquent de Laravel, ni con ninguna tecnología adicional como MySQL, SQlite, etc.

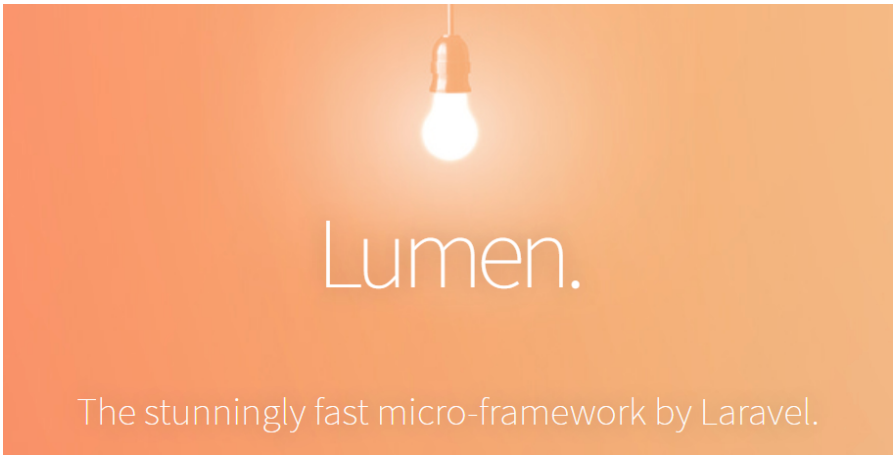


Figura 4.9.: Logo de Lumen.

4.2.2. Visual Studio Code

Visual Studio Code[11] es un editor de texto ligero y potente creado por Microsoft en Noviembre de 2015. Dispone de muchos plugins o extenxiones[12] mediante las cuales añadir mejoras al editor, como podría ser mejorar el aspecto visual del editor, cambiar la tipografía, corregir errores en la sintaxis PHP, autocompilación, etc.

Por lo tanto, dada su versatilidad y facilidad de uso, esta ha sido la herramienta utilizada para el desarrollo del código de la parte del servidor del proyecto.

[10]Web de Lumen. `https://lumen.laravel.com/docs/5.7`
[11]Web de Visual Studio Code. `https://code.visualstudio.com/`
[12]Tienda de Extensiones. `https://marketplace.visualstudio.com/`

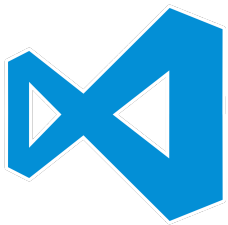


Figura 4.10.: Logo de Visual Studio Code.

4.2.3. Swagger

Para consumir una API Rest necesitamos saber qué peticiones acepta y qué parámetros contienen dichas peticiones, para que el desarrollador sepa qué puede obtener de dicha API. Para este propósito utilizaré Swagger[13] (Lenguaje de Modelado de API RESTful) que nos ayudará a crear un esquema de la aplicación.

Swagger es un lenguaje de modelado de datos que utiliza el formato YAML[14], que nos permite definir todas las propiedades que tendrá la petición HTTP.

Figura 4.11.: Logo de Swagger.

4.3. Partes comunes (Servidor y Cliente)

Para mantener el sistema de control de versiones con Git en un repositorio remoto he decidido utilizar el servicio GitHub[15]. Con GitHub podemos mantener una copia de seguridad en remoto además de poder compartirla entre diferentes ordenadores. Me decidí por este servicio por la sencillez de crear, revertir y modificar commits, aparte de las herramientas internas que ofrece (Wiki, Issues...), además de que tenemos la posibilidad de crear repositorios privados ilimitados[16].

[13]Web de Swagger. `https://swagger.io/`
[14]YAML en la Wikipedia. `https://es.wikipedia.org/wiki/YAML`
[15]Web de GitHub. `https://github.com`
[16]Precios GitHub. `https://github.com/pricing`

A través de la web de GitHub podemos acceder a una sección donde tendremos un tablero Kanban (ver imagen 4.12), como ya comentamos en anteriores apartados, y que nos ayuda a tener una imagen del estado actual del proyecto (ver sección 3). Existe la posibilidad de automatización (ver imagen 4.13) del tablero que incorpora GitHub, gracias a esto, se van añadiendo de forma automática todos los issues nuevos (ver sección 3.2) en la columna "To do". Se moverán de forma manual los issues a la columna "In progress" cuando se comience con su desarrollo y a "Done" cuando se haya completado.

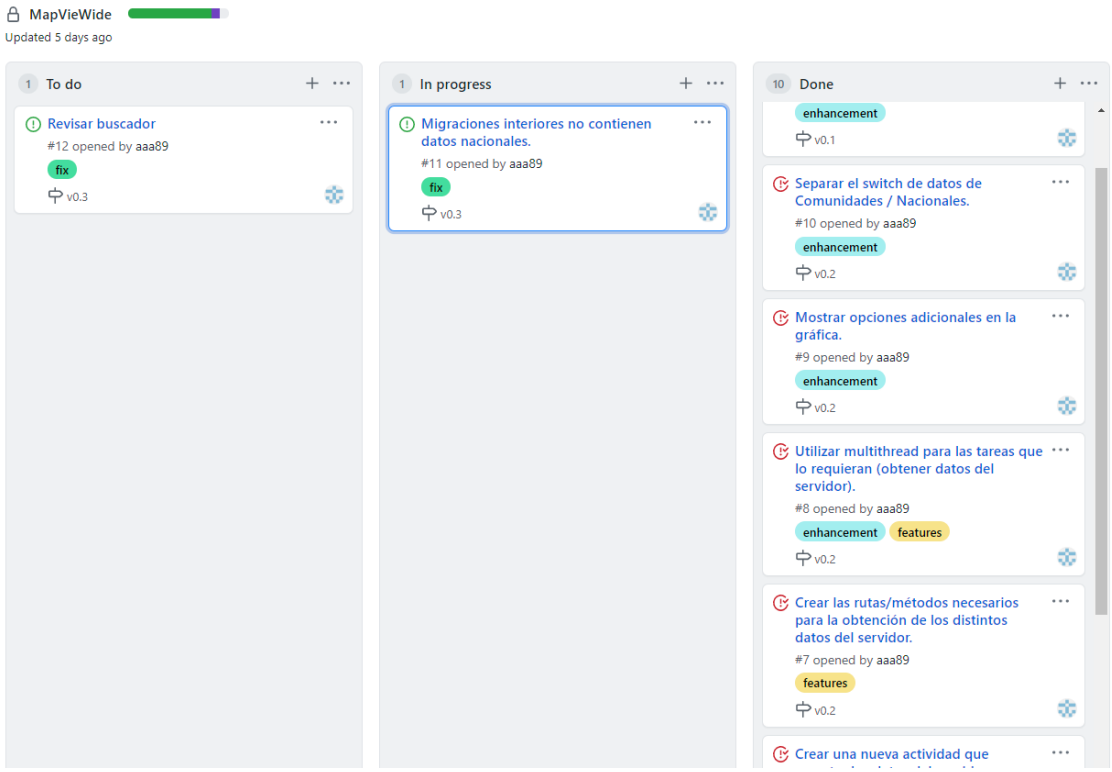


Figura 4.12.: Tablero Kanban del proyecto.

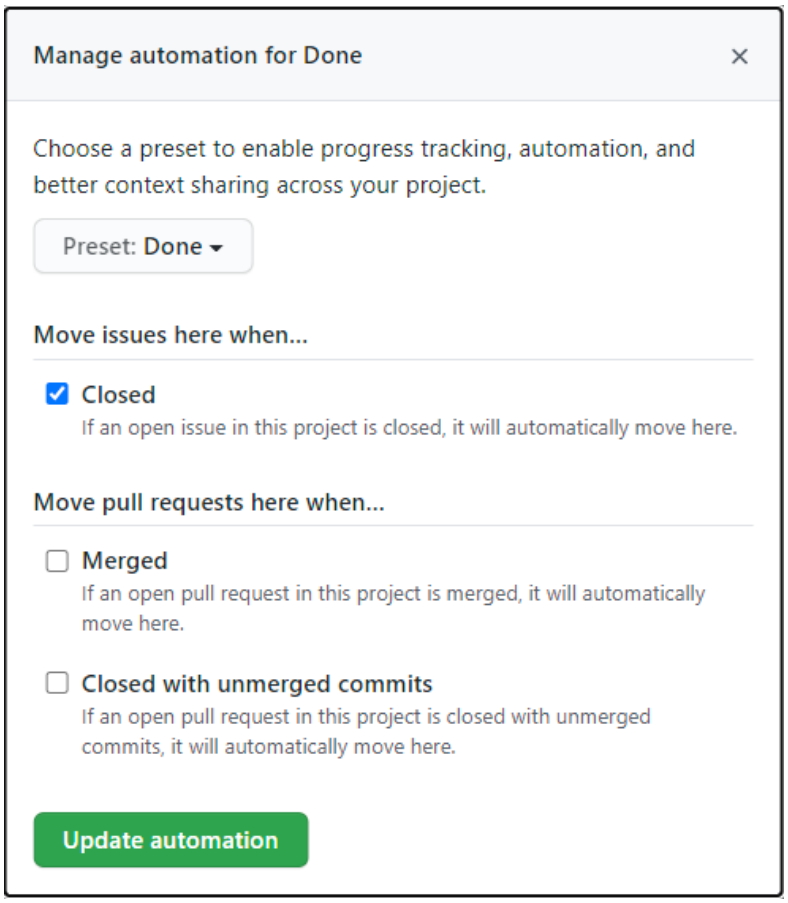


Figura 4.13.: Configuración de reglas de la columna Done.

Otra herramienta interesante que contiene GitHub es su publicación de versiones (ver imagen 4.14). Mediante esta publicación podemos añadir una descripción con las nuevas funcionalidades añadidas o errores arreglados que contenga la versión. En el momento de crear la publicación, se enlaza con un commit y se crean dos ficheros comprimidos con el código fuente de dicho commit.

Además del código fuente, se pueden añadir otros elementos a la publicación, como podría ser el fichero APK generado para instalar Android o cualquier otro fichero o documento que nos parezca oportuno añadir.

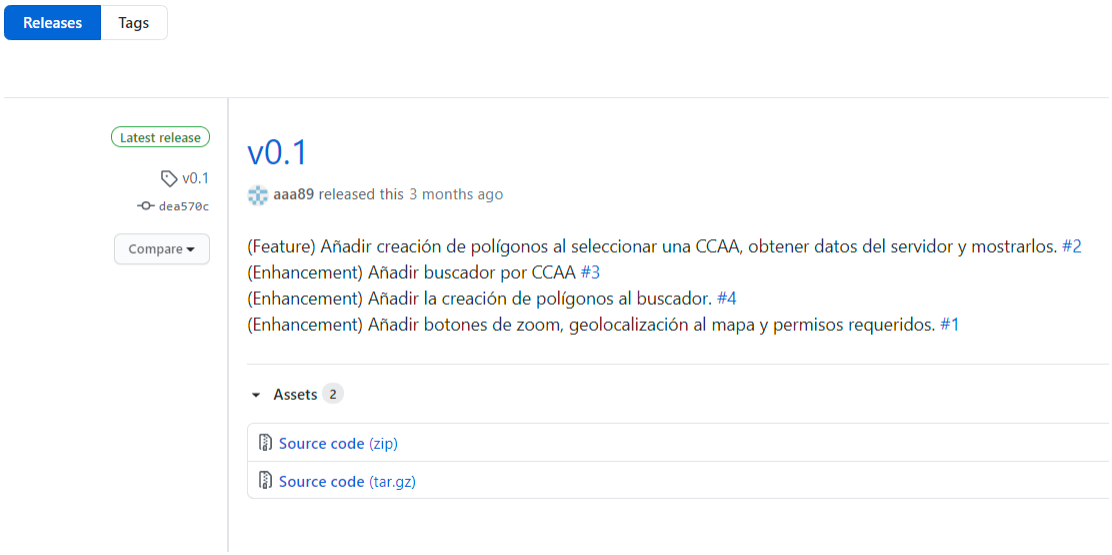


Figura 4.14.: Sección de publicaciones de versiones de GitHub.

A la hora de unir dos ramas de nuestro sistema de control de versiones (ver imagen 3.1) se puede hacer mediante pull request que es una forma más sencilla de realizar los merges que nos ofrece Git, además de indicarnos si existe algún problema durante el mergeo.

Estas solicitudes permiten tener una versión del código en la que está unido el código pero todavía no se ha subido, para que nos resulte más sencillo de visualizar el merge, además de tener la posibilidad de realizar integración continua mediante alguna herramienta como Travis[17].

Figura 4.15.: Logos de Git y GitHub

[17]Web de Travis. `https://travis-ci.org/`

4.3.1. GitKraken

Utilizar Git mediante la línea de comandos puede llegar a ser repetitivo y tedioso, por lo tanto decidí utilizar un programa que me ayudase con ello a través de una interfaz gráfica, lo que simplifica el trabajo.

GitKraken[18] fue creada en agosto de 2014 y ya cuenta con más de un millón de usuarios. Algunas de las funcionalidades por las que me interesó utilizar GitKraken fueron las siguientes: la integración con GitHub, es multiplataforma (Windows, MacOS...) y también porque cuenta con una interfaz bastante intuitiva y sencilla de utilizar.

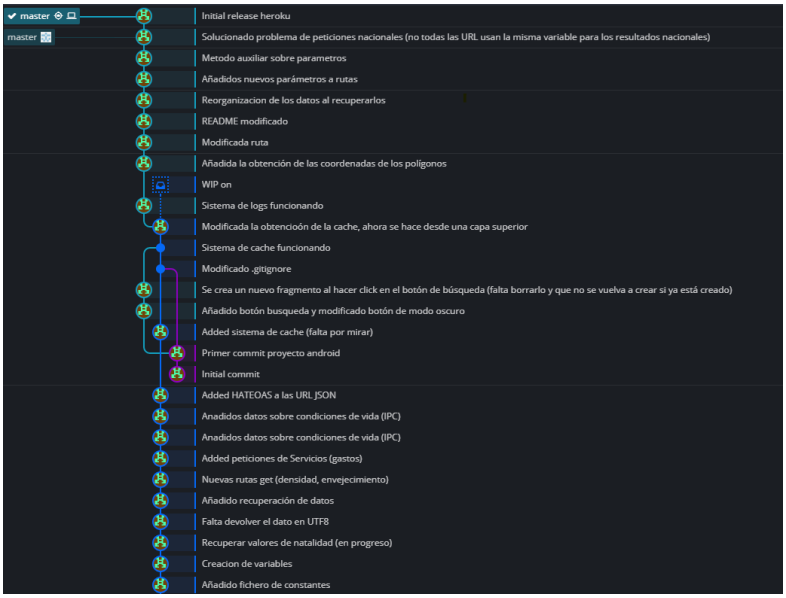


Figura 4.16.: Aplicación GitKraken.

Figura 4.17.: Logo de GitKraken.

[18]Web de GitKraken. `https://www.gitkraken.com/`

4.3.2. Draw.io

En la fase de análisis inicial del proyecto y durante la fase de análisis de cada tarea, es necesaria una herramienta que nos permita crear un diseño, mediante distintos tipos de diagramas, para poder hacernos una mínima idea de como estará estructurado nuestro proyecto. Tendremos la posibilidad también de descubrir posibles fallos que puedan surgir antes de empezar de lleno con la implementación.

La herramienta utilizada para esta problemática ha sido Draw.io[19]. La cual cuenta con las siguientes funcionalidades:

- Gran librería de figuras.
- Los elementos que forman los diagramas con personalizables.
- Permite almacenar los diagramas en la nube y poder trabajar con ellos de manera simultánea.

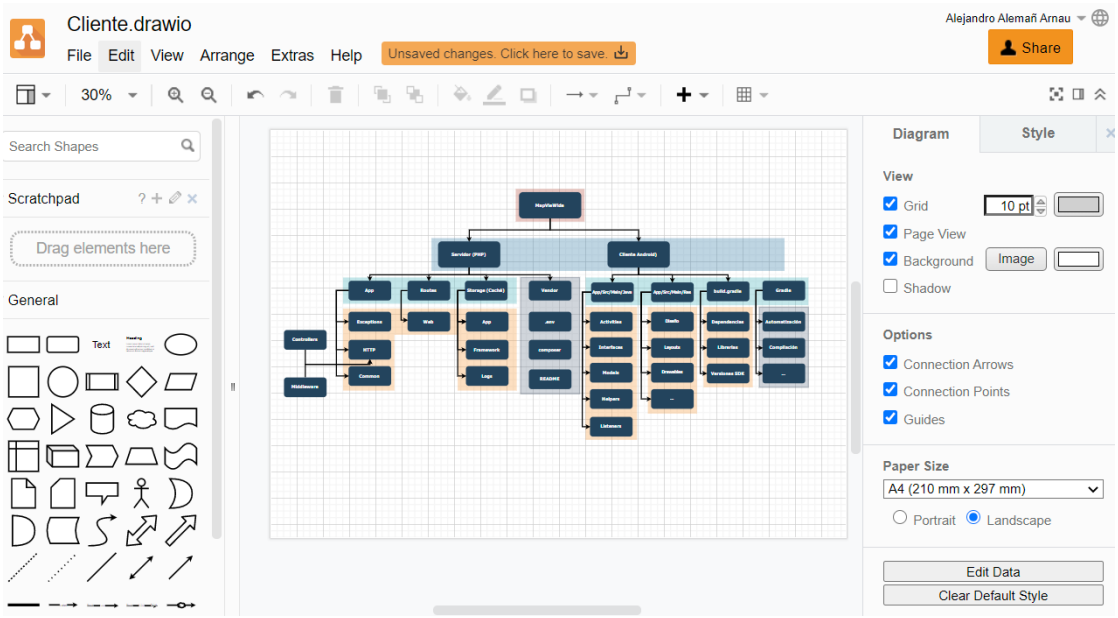


Figura 4.18.: Herramientas de diagramas Draw.io.

[19]Web de Draw.io. `https://www.draw.io/`

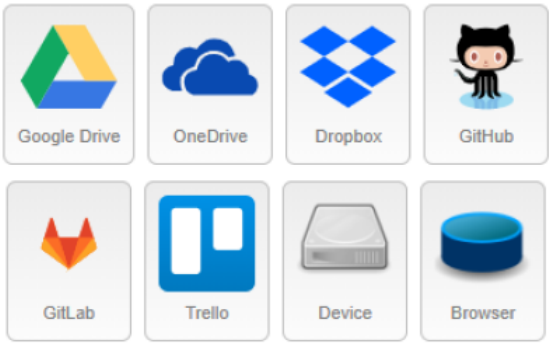


Figura 4.19.: Servicios de almacenamiento en la nube de Draw.io.

4.3.3. LaTeX y Overleaf

LaTeX[20] es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Para la memoria de este proyecto decidí utilizar esta tecnología por recomendación del tutor, además de otras recomendaciones de compañeros de la universidad.

Para escribir el código LaTeX y compilar el proyecto utilicé Overleaf, una herramienta online que facilita mucho el trabajo con esta tecnología. Entre sus ventajas cabe destacar la incorporación de plantillas para la escritura de distintos tipos de documentos, la integración de todas las librerías y paquetes necesarios para compilar los proyectos (sin tener que instalar nada), y la posibilidad de trabajar de forma colaborativa con otros usuarios sobre un mismo proyecto LaTeX.

[20]Web de LaTeX. `https://www.latex-project.org/get/`

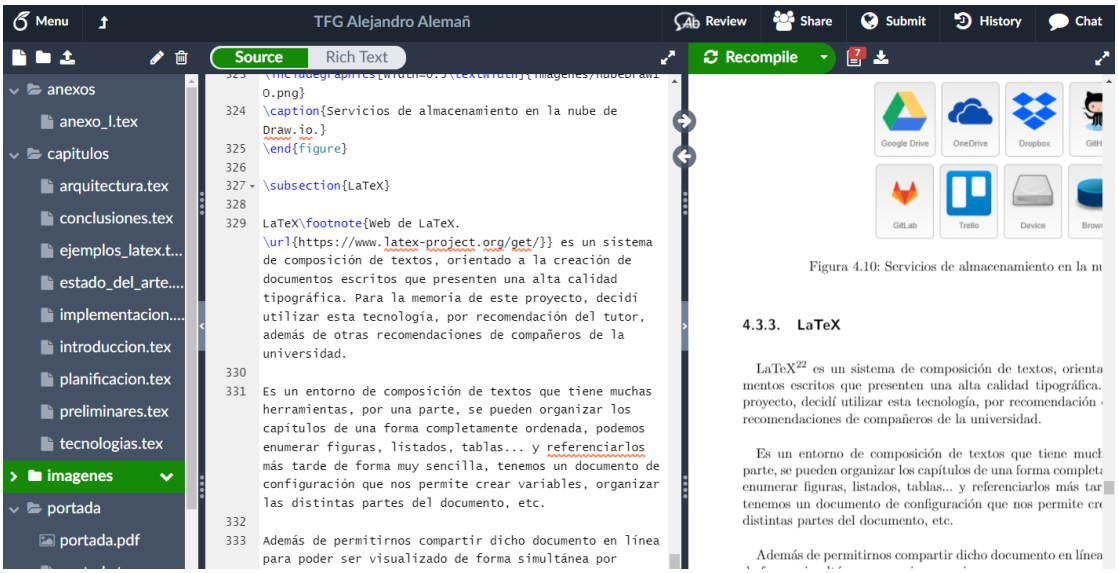


Figura 4.20.: Herramienta Overleaf.

Figura 4.21.: Logo de LaTeX.

5. Arquitectura

Tener una arquitectura clara de cara al acceso a los datos y a las funciones es muy importante en el desarrollo. Esto nos ahorrará tiempo y mejorará la escalabilidad del proyecto, solucionará o mermará algunos errores, refactorizaciones de código futuras y duplicación de código que podríamos tener si no tenemos definida la arquitectura del proyecto, lo que lo haría insostenible.

En primer lugar veremos la arquitectura utilizada en el servidor, que vendrá condicionada por el framework utilizado (ver sección 4.2.1) y a continuación por la utilizada en el cliente, que, al no tener una estructura definida, Android no obliga a seguir ningún patrón de software, deberemos de diseñarla nosotros a nuestro parecer.

5.1. Arquitectura servidor

Al utilizar el framwork Lumen estamos condicionados a la arquitectura que nos define en propio framwork. En este caso Laravel define la arquitectura de MVC (Modelo-Vista-Controlador), pero al usar Lumen, que es una versión reducida destinada al desarrollo de una API, la vista se elimina, por lo tanto nos quedarían solamente el modelo y el controlador.

En el modelo se suelen definir las entidades de la base de datos y como se relacionan entre sí. En nuestro caso no tenemos base de datos, sino únicamente llamadas cacheadas a las distintas rutas de la aplicación, por lo tanto nos queda el controlador, que es el que consulta los modelos y crea la respuesta a las llamadas de la API.

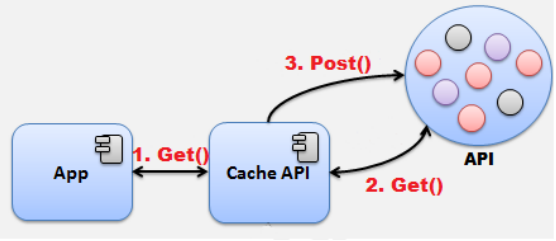


Figura 5.1.: Esquema de datos del servidor.

5.2. Arquitectura cliente

Para la implementación de la aplicación cliente se ha utilizado la arquitectura MVVM (Model-View-ViewModel) (ver imagen A.4).

Los principales elementos que componen esta arquitectura son los siguientes:

- **Vista**: la vista es sencilla ya que no contiene lógica, solo tiene las funciones de modificar los elementos de la vista según ordene el viewModel y de pasar las interacciones de la vista al viewModel para que este las procese y actúe en consecuencia.

- **ViewModel**: la principal tarea del viewModel es ejecutar toda la lógica de una pantalla. También tiene la función de llamar a servicios REST o de extraer datos de la base de datos. Una vista debe tener un viewModel pero puede ser compartido entre varias, la comunicación con la vista se realiza mediante objetos LiveData (ver sección 6.2.2).

- **Repositorio**: los viewModels hacen llamadas a los repositorios para la obtención de los datos de la API de forma abstracta o bien los datos de la base de datos local. Los repositorios son los encargados de obtener los datos de la base de datos (Room). En nuestro caso, esto no lo usaremos ya que no tenemos base de datos, únicamente obtendremos los datos de la API (Retrofit) y los trataremos para ser devueltos al viewModel.

- **Modelo**: para el acceso a la base de datos local se utiliza Room a través de interfaces DAO, pero nosotros no lo utilizaremos dado que no tenemos base de datos, pero sí tendremos los modelos necesarios para utilizar Retrofit.

- **Servicio Remoto (Retrofit)**: el acceso a la API es controlado por la librería Retrofit, para la cual crearemos una/s interfaz/ces donde estableceremos las llamadas que se pueden realizar al API, como explicamos en la sección anterior 4.1.3.2.

En la siguiente imagen se puede ver cómo se estructuran los elementos en el modelo MVVM según la documentación de android con algunas modificaciones:

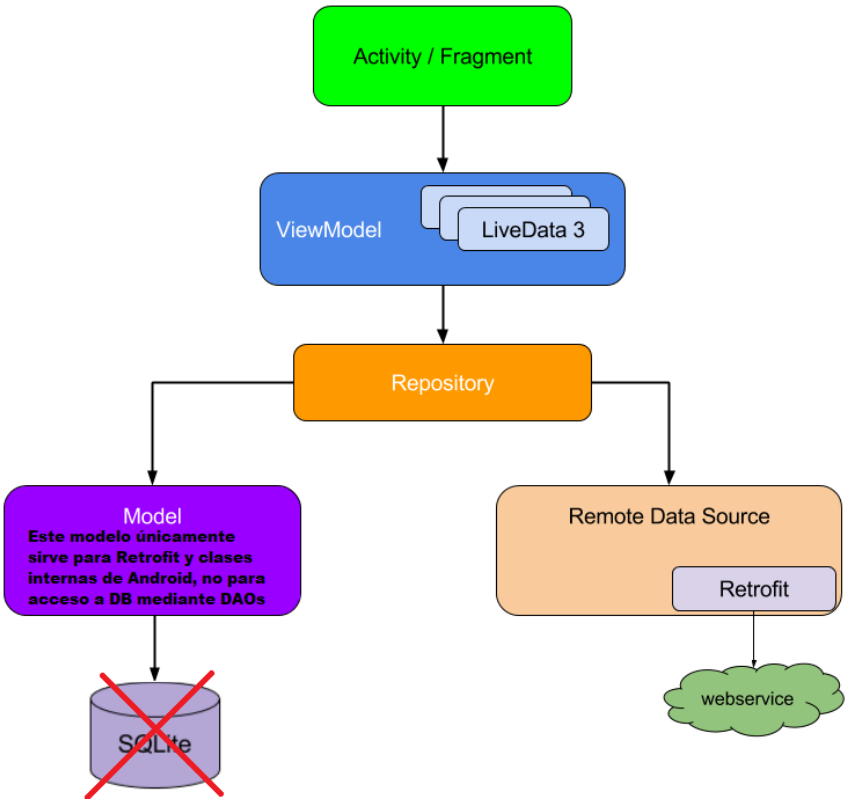
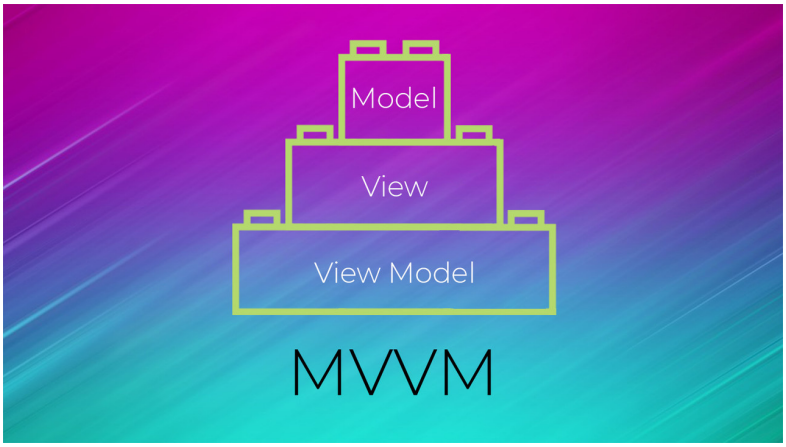


Figura 5.2.: Arquitectura seguida en el cliente Android (MVVM).



En el siguiente diagrama de clases (ver imagen 5.3) podemos ver como están divididas las capas de la aplicación y cómo se relacionan entre sí:

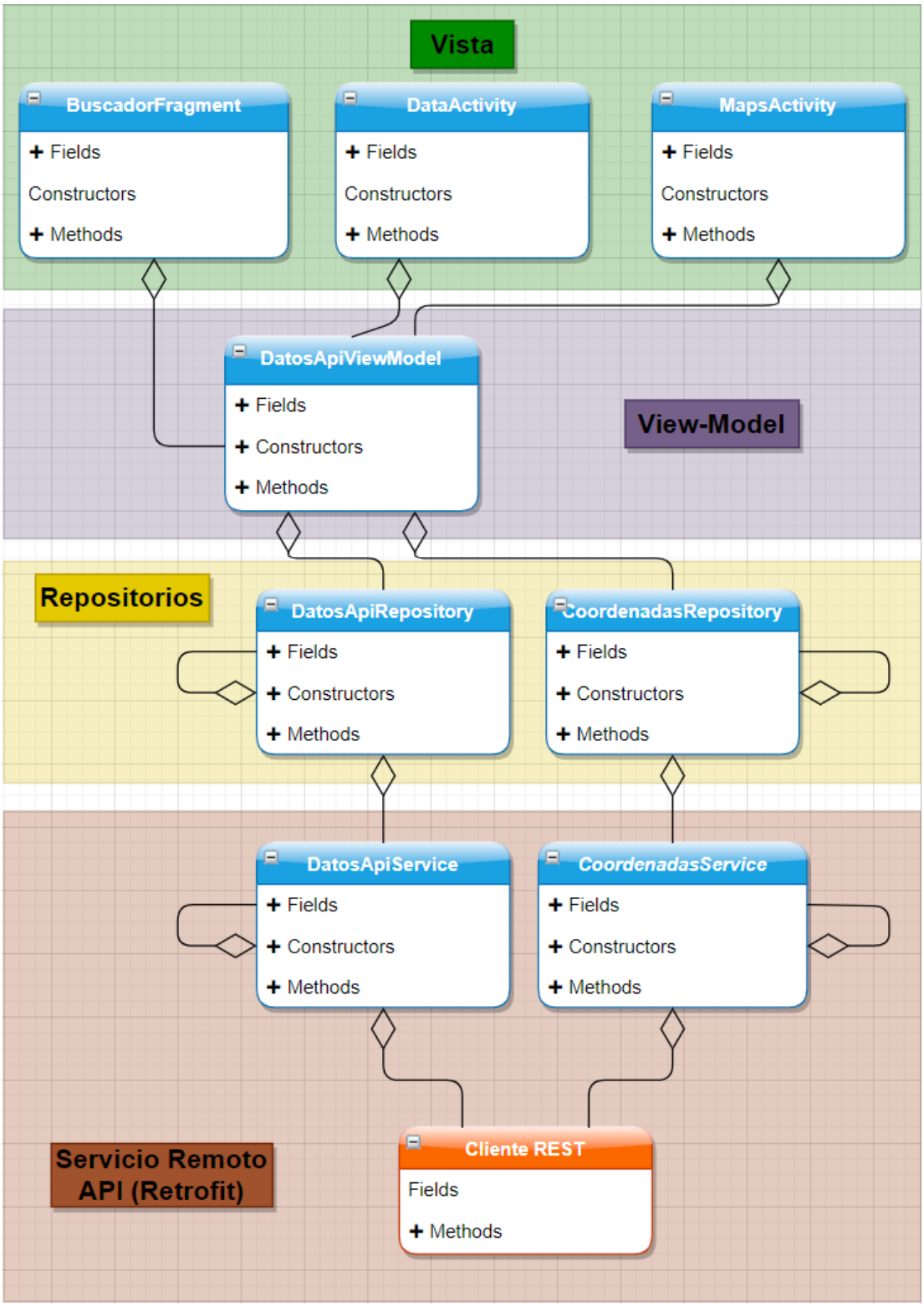


Figura 5.3.: Diagrama de clases del cliente Android.

6. Implementación

6.1. Implementación del servidor

En primer lugar hablaremos sobre el framework utilizado en el servidor, en este caso Lumen. Uno de los ficheros principales de este framework es "web.php", el cual contendrá todas las rutas posibles de nuestra API. En este framework, al realizar las llamadas a una ruta de la API, inicialmente se ejecuta un middleware que proporciona un mecanismo conveniente para filtrar las solicitudes HTTP que ingresan en la aplicación.

En cada ruta se pueden usar parámetros opcionales, los cuales están representados mediante las llaves "{}" como podemos observar en el listado 6.10, por ejemplo con {idCCAA}. Estos parámetros serán recibidos por el método que se muestra.

Como podemos observar en el listado 6.1, tenemos una petición GET a la ruta /poblacionCCAA/{idCCAA}[/{periodo}, todas las llamadas GET y POST de la API serán validadas por los middlewares que se comentarán en el siguiente apartado.

A grosso modo, un middleware es una capa de la aplicación por la que deben pasar todas las peticiones que lo estén utilizando. Esto nos permitirá realizar acciones antes de que se ejecute la petición y también nos permitirá filtrar las peticiones que se vayan a realizar.

Listado 6.1: Un método GET del API.

```
1 $router->get('poblacionCCAA/{idCCAA}[/{periodo}]', function (
      $periodo = null, $idCCAA = null) {
2     $url = "https://servicios.ine.es/wstempus/js/ES/DATOS_TABLA
          /2853";
3
4     $url = addParametersUrl($url, $periodo, $idCCAA);
5     return getResponseJson(getJson($url, $periodo));
6 });
```

6.1.1. Middleware

En nuestro caso tenemos dos middlewares, el primero define las reglas para cada petición (CORS[1]), pero únicamente nos hemos fijado en la cabecera de cada petición y el método que se está utilizando (GET, POST...).

Listado 6.2: Ejemplo de implementación del middleware Cors.

```
public function handle($request, Closure $next)
    {
        $headers = [
            'Access-Control-Allow-Origin'      => '*',
            'Access-Control-Allow-Methods'     => 'POST, GET',
            'Access-Control-Allow-Headers'     => 'Content-Type,
               Authorization, X-Requested-With'
        ];

        $response = $next($request);
        foreach ($headers as $key => $value) {
            $response->header($key, $value);
        }
        return $response;
    }
```

Por otra parte, tenemos un segundo middleware que añade a las llamadas realizadas a la API la información que nosotros le indiquemos. La implementación sería la siguiente:

Listado 6.3: Ejemplo de implementación del middleware ResponseWrapper.

```
public function handle($request, Closure $next) {
    $response = $next($request);
    if ($response instanceof JsonResource) {
        if (!isset($response->getData()->status)) {
            $newResponseData['status'] = 'success';
            $newResponseData['code'] = $response->getStatusCode();
            $newResponseData['data'] = $response->getData();
            $response->setData($newResponseData);
        }
    }
    return $response;
}
```

[1]CORS en Wikipedia. https://es.wikipedia.org/wiki/Intercambio_de_recursos_de_origen_cruzado

6.1.2. Cacheado de peticiones

Lumen nos permite coger funcionalidades de Laravel, en este caso, el sistema de cacheado[2]. Nos proporciona una API expresiva y unificada para varios backends de almacenamiento en caché. Laravel admite backends populares de almacenamiento en caché como Memcached y Redis que están listos para ser implementados.

He decidido utilizar el propio sistema de cacheado que nos ofrece Laravel, que para este proyecto es suficiente. En caso de ampliar el proyecto y necesitar un mayor performance, tendríamos que haber optado por alguna de las dos opciones mencionadas anteriormente.

Listado 6.4: Ejemplo de uso del método Cache::put sobre la caché.

```
function getResponseJson($jsonData) {
    $i = 0;
    $ttl = 600;
    $mapData = array('results' => [], 'href' => $_SERVER["
       REQUEST_URI"]);

    foreach ($jsonData as $comunidad) {
        array_push($mapData['results'], array('categoria' =>
           $comunidad['Nombre'], 'datos' => []));
        foreach ($comunidad['Data'] as $value) {
            if ($value['Valor'] > 0) {
                array_push($mapData['results'][$i]['datos'], array
                   ('year' => $value['Anyo'], 'valor' => $value['
                   Valor']));
            }
        }
        //Eliminamos los datos vacíos
        if (empty($mapData['results'][$i]['datos'])) {
            unset($mapData['results'][$i]);
        }
        $i++;
    }

    $newResults = array_values($mapData['results']);
    $mapData = array('results' => $newResults, 'href' => $_SERVER
       ["REQUEST_URI"]);

    //Añadida clave de la caché para más tarde poder recuperarla.
    Cache::put($_SERVER["REQUEST_URI"], $mapData, $ttl);

    return $mapData;
}
```

[2]Documentación Laravel. `https://laravel.com/docs/7.x/cache`

Como podemos observar en la línea 24, se está cacheando mediante el método **Cache::put** el dato ("$mapData") que se obtiene de las operaciones del método con la clave "$_SERVER['REQUEST_URI']", que es la llamada realizada por el usuario al API, y por lo tanto única. Por último tenemos la variable "$ttl" que contiene el tiempo de vida de la caché, cada cuanto queremos que se refresquen los datos.

Para comprobar si existe la caché en el sistema y que podamos recuperarla, se hace uso del método **Cache::has**, como se puede observar en la línea 7, que comprueba si existe el valor que le pasamos en la caché y, en caso de ser así, lo devolvemos como se puede observar en la línea 10, para evitar que el servidor haga una petición al API y consuma recursos adicionales.

Otra cosa importante a comentar es el uso del método **isset()**, el cual verifica si una variable está establecida, lo que significa que debe estar declarada y no ser NULL. En este caso nos sirve para cuando se utiliza la aplicación de forma remota.

Listado 6.5: Ejemplo de uso del método Cache::has y Cache::get sobre la caché.

```
1  public static function call($container, $callback, array
       $parameters = [], $defaultMethod = null){
2      if (static::isCallableWithAtSign($callback) || $defaultMethod)
           {
3          return static::callClass($container, $callback,
               $parameters, $defaultMethod);
4      }
5
6      return static::callBoundMethod($container, $callback, function
           () use ($container, $callback, $parameters) {
7          if (Cache::has(isset($_SERVER["REQUEST_URI"]))) {
8              Log::debug("Se ha obtenido de la caché la petición " .
                   $_SERVER["REQUEST_METHOD"] . " con la ruta " .
                  $_SERVER["REQUEST_URI"], ['tipo' => $_SERVER["
                  REQUEST_METHOD"], 'url' => $_SERVER["REQUEST_URI
                  "]]);
9              return Cache::get(isset($_SERVER["REQUEST_URI"]));
10         } else {
11             Log::debug("Se ha realizado una petición " . isset(
                   $_SERVER["REQUEST_METHOD"]) . " con la ruta " .
                   isset($_SERVER["REQUEST_URI"]), ['tipo' => isset(
                   $_SERVER["REQUEST_METHOD"]), 'url' => isset(
                   $_SERVER["REQUEST_URI"])]);
12             return call_user_func_array(
13                 $callback,
14                 static::getMethodDependencies($container,
                       $callback, $parameters)
15             );
16         }
```

6.1.3. Logs

Para el sistema de logs de la aplicación se ha utilizado la herramienta que nos proporciona Lumen, en este caso utiliza una capa superior de la biblioteca Monolog. En un principio Lumen está configurado para crear un único archivo de registro para la aplicación, pero esto puede ser modificado.

Existen diversos tipos de Logs: de alerta, crítico, de información, de debug, etc. En mi caso he utilizado el de debug para el desarrollo de la aplicación, que es el que más información nos muestra, por si ocurre algún error, poder saber de donde viene con exactitud y solucionarlo.

Aquí tenemos un ejemplo del uso de los logs en el código, donde indicamos con qué método HTTP se está realizando la petición y con qué URL ha llamado el usuario al API:

Listado 6.6: Ejemplo del uso de Logs con uso de caché.

```
1 //Petición que pasa por la caché
2 Log::debug("Se ha obtenido de la caché la petición " . $_SERVER["
    REQUEST_METHOD"] . " con la ruta " . $_SERVER["REQUEST_URI"],
    ['tipo' => $_SERVER["REQUEST_METHOD"], 'url' => $_SERVER["
    REQUEST_URI"]]);
```

```
610 [2020-07-28 09:05:05] local.DEBUG: Se ha realizado una petición POST con la ruta /area/349044 {"tipo":"POST","url":"/area/349044"}
611 [2020-07-28 09:05:41] local.DEBUG: Se ha realizado una petición POST con la ruta /area/349036 {"tipo":"POST","url":"/area/349036"}
612 [2020-07-28 09:07:10] local.DEBUG: Se ha obtenido de la caché la petición POST con la ruta /area/349055 {"tipo":"POST","url":"/area/349055"}
613 [2020-07-28 09:07:17] local.DEBUG: Se ha realizado una petición POST con la ruta /area/349036 {"tipo":"POST","url":"/area/349036"}
614 [2020-07-28 09:07:52] local.DEBUG: Se ha realizado una petición POST con la ruta /area/349052 {"tipo":"POST","url":"/area/349052"}
615 [2020-07-28 09:07:55] local.DEBUG: Se ha obtenido de la caché la petición POST con la ruta /area/349036 {"tipo":"POST","url":"/area/349036"}
616 [2020-07-28 09:08:04] local.DEBUG: Se ha realizado una petición POST con la ruta /area/348981 {"tipo":"POST","url":"/area/348981"}
617 [2020-07-28 09:08:09] local.DEBUG: Se ha realizado una petición POST con la ruta /area/1154757 {"tipo":"POST","url":"/area/1154757"}
618 [2020-07-28 09:08:16] local.DEBUG: Se ha obtenido de la caché la petición POST con la ruta /area/349036 {"tipo":"POST","url":"/area/349036"}
619 [2020-07-28 09:08:19] local.DEBUG: Se ha obtenido de la caché la petición POST con la ruta /area/349044 {"tipo":"POST","url":"/area/349044"}
620 [2020-07-28 09:08:23] local.DEBUG: Se ha realizado una petición POST con la ruta /area/1154756 {"tipo":"POST","url":"/area/1154756"}
```

Figura 6.1.: Ejemplo de logs almacenados en el directorio.

6.1.4. Documentación de la API

Para utilizar la API desde el cliente hace falta tener una documentación donde se definan qué tipo de peticiones puede recibir, qué devuelve y para qué sirven.

Para todo este propósito he utilizado la herramienta RAML (RESTful API Modeling Language), un modelo de datos con el formato YAML que nos permite establecer la documentación de una API indicando con el detalle que necesitemos el formato de las peticiones, las respuestas que devuelve, los tipos de errores, etc.

Aquí podemos ver un ejemplo de la documentación de un método de nuestra API, donde se realiza una petición HTTP de tipo GET a la URL /poblacionCCAA. Esta petición contiene query parameters (parámetros de la query) que se necesitan para realizar de forma correcta la llamada a dicha URL. En este caso tenemos dos, uno obligatorio y otro opcional, por último se definen las llamadas que puede devolver, en nuestro caso, un código 200 que significa que todo ha ido bien, código 500, que significa que hay un error interno del servidor, y un 404 que significa que no se ha encontrado dicha URL. Estos códigos de error son los definidos en el middleware ResponseWrapper que se habló en apartados anteriores (ver sección 6.1.1):

Listado 6.7: Ejemplo de un método documentado mediante Swagger.

```
/area/{id}:
    post:
      tags:
      - "Área mapa"
      summary: "Obtenemos el respectivo área en puntos"
      operationId: "area/id"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - name: "id"
        in: "path"
        description: "ID de la CCAA deseada"
        required: true
        type: "string"
      responses:
        "200":
          description: "Operación realizada correctamente"
        "404":
          description: "Dirección no encontrada, revisar URL"
        "500":
          description: "Error interno en el servidor, revisar URL"
```

Para ver la documentación completa ver el anexo B.

6.1.5. Errores en peticiones HTTP

La gestión de los errores de las llamadas al API se ha realizado utilizando el código de estado de la petición HTTP[3] y añadiendo mensajes al cuerpo de la respuesta del error.

Cuando ocurre un error en el servidor se devuelve el estado HTTP correspondiente y el cuerpo en formato JSON con los campos correspondientes indicando el error obtenido.

6.2. Cliente

A continuación vamos a ver la implementación del cliente en Android. Dado que Android no obliga al uso de ningún patrón software, el código de la aplicación ha sido estructurado siguiendo el patrón MVVM, como se ha explicado anteriormente. Más adelante se puede observar el resultado final de la aplicación (Anexo A).

6.2.1. Interfaz

Para la creación de la interfaz de nuestra aplicación se han usado plantillas por defecto de los distintos items de la aplicación (Spinners, Textbox...) que nos ofrece la documentación de Android.

Existen guías de diseño que nos ayudan con el diseño de la aplicación, pero al no tener tantas pantallas, decidí hacerlo de forma sencilla. Un ejemplo de estas guías de diseño podría ser las Material Design[4] creadas por Google, normalmente para cada versión de Android existe una distinta o actualizaciones de las mismas.

Para la creación de las interfaces Android Studio (Sección 4.1.2) nos aporta un editor gráfico donde podemos añadir los componentes que deseemos bien mediante XML, o bien desde la parte gráfica donde podremos añadir los componentes arrastrándolos donde nos interese (ver imágenenes 6.2 y 6.3).

Después de haber realizado los oportunos cambios en la interfaz gráfica, esta será cargada desde un fragmento (Fragment) y se podrán obtener dichos elementos de manera programática haciendo uso del método “findViewById(id)”, el cual nos devuelve el elemento del interfaz que tiene dicho id.

[3]Wikipedia. `https://developer.mozilla.org/es/docs/Web/HTTP/Status`
[4]Web de Material Design. `https://material.io/design/`

Para nuestra aplicación hemos creado dos actividades y un fragmento:

- Primera actividad (Mapa): Contiene los items (botones, mensajes de información, etc.) para obtener los polígonos de las distintas comunidades autónomas y también la posibilidad de acceder a la segunda actividad.

- Segunda actividad (Menú): Contiene los elementos que nos permitirán recuperar la información del servidor (Spinners, Checkbox...) y donde mediante una gráfica se mostrarán todos los datos que se recuperen del API.

- Fragmento (Buscador): También se ha creado un fragmento para el uso del buscador, ya que es una utilidad que se encuentra dentro de la actividad del mapa y se necesita mostrar de forma aislada por las búsquedas y operaciones que realiza.

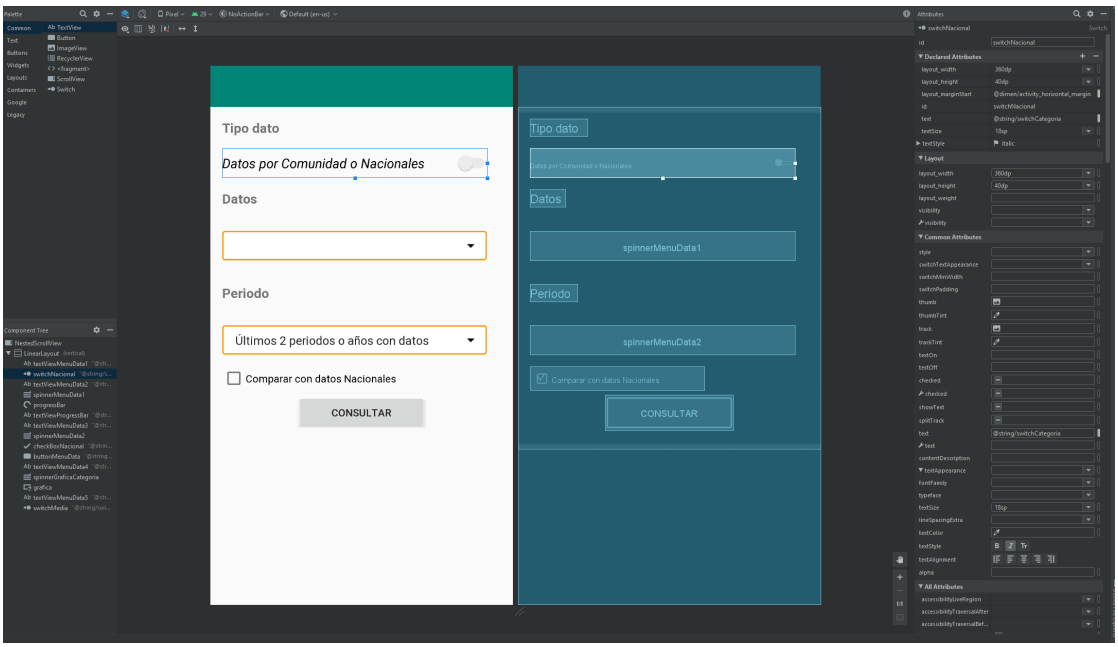


Figura 6.2.: Herramienta gráfica para desarrollar interfaces en Android Studio.

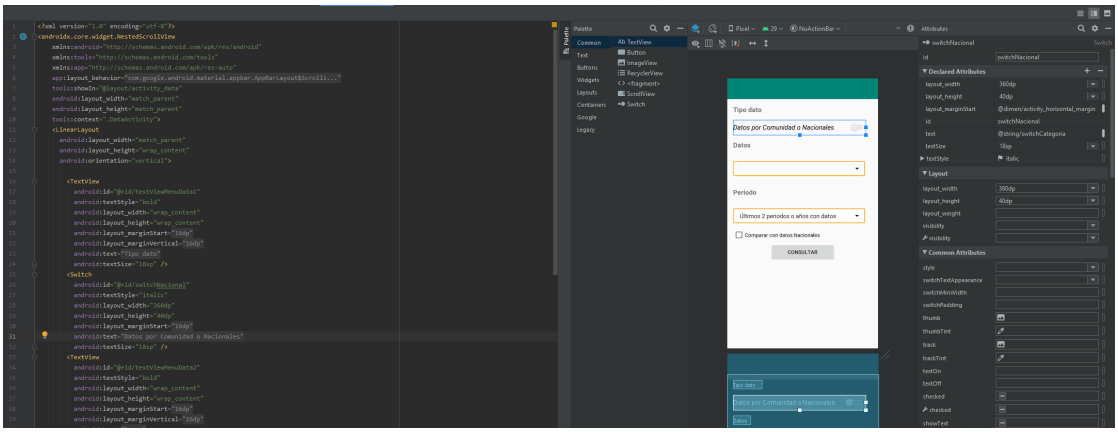


Figura 6.3.: Herramienta gráfica de Android studio en vista XML y gráfica.

6.2.2. LiveData

A la hora de mostrar la interfaz al usuario deben de cargarse los datos para que el usuario pueda utilizarlos. Si ocupamos el hilo de ejecución con esta tarea, la aplicación quedaría congelada hasta que recibiera los datos. Para evitar esto hace falta cargar los datos de forma asíncrona como se puede observar en la imagen 6.4 y utilizar los objetos LiveData, más concretamente los MutableLiveData, que nos permitirán modificar los datos, también utilizar distintos hilos de ejecución dentro de la aplicación.

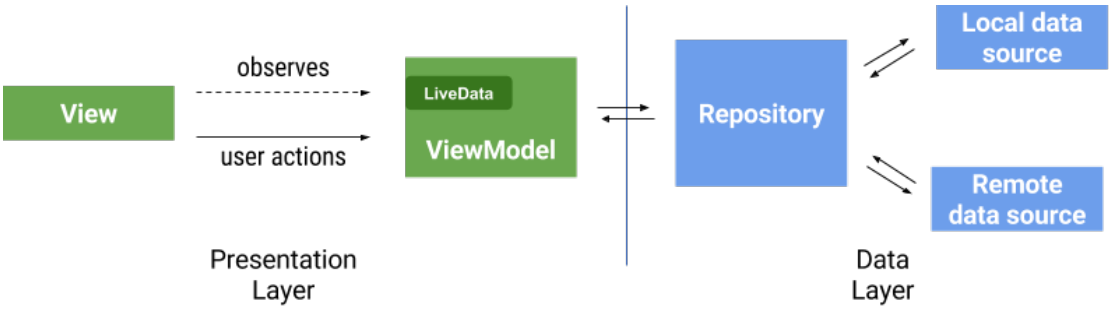


Figura 6.4.: Carga asíncrona de datos usando un LiveData.

Como podemos observar en el siguiente listado, se hace la llamada al método "getAutoCompletePrediction" del ViewModel indicando que cuando se reciban los datos se llamará al método setAutoCompletePrediction, el cual recibirá como parámetro la query del autoCompletePrediction y establecerá los valores necesarios en la vista.

Listado 6.8: Ejemplo del uso de objetos LiveData.

```
viewModel.getAutoCompletePrediction(place).observe(this, this::
    setAutoCompletePrediction);
```

En el ViewModel debemos de crear el objeto MutableLiveData con el tipo de dato que debemos devolver y de forma asíncrona hacer una llamada al repositorio como se puede observar en el siguiente listado.

Devolveríamos el objeto MutableLiveData y cuando tengamos el valor de AutoCompletePrediction, lo enviaremos a través del objeto MutableLiveData que habíamos devuelto anteriormente.

Listado 6.9: Ejemplo de uso del objeto MutableLiveData desde el ViewModel.

```
    public MutableLiveData<AutoCompletePrediction>
        getAutoCompletePrediction(Place place){
        MutableLiveData<AutoCompletePrediction> mutableLiveData =
            new MutableLiveData<>();

        new Thread(() -> {
            try{
                place = place.getAddress();
            }
            catch (IOException e) {
                e.printStackTrace();
                place = null;
            }
            mutableLiveData.postValue(place);
        }).start();

        return mutableLiveData;
    }
```

6.2.3. Retrofit

Para la comunicación con el servidor accedemos a la API mediante peticiones HTTP que nos permitan obtener los datos. He utilizado la librería de Retrofit (ver sección 4.1.3.2) mediante la que podemos realizar peticiones de forma muy sencilla, abstrayéndonos así de dicha tarea.

La librearía Retrofit se basa en una interfaz de Java en la cual tenemos que definir tantos métodos como peticiones realice el API, y debe recibir los mismos parámetros, al igual que devolver el mismo JSON, con los mismos nombres.

A continuación veremos unos cuantos ejemplos de métodos (llamadas) que tendríamos en una interfaz donde se utiliza Retrofit:

Listado 6.10: Ejemplo de llamadas en la interfaz Retrofit.

```
@GET("poblacionCCAA/{idCCAA}/{periodo}")
  Call<MapVieWideResponseDatos> getPoblacionCCAA(@Path("idCCAA")
     String idCCAA, @Path("periodo") String periodo);
@GET("preciosTarifasApartamentosTuristicosNacional/{periodo}")
  Call<MapVieWideResponseDatos>
     getPreciosTarifasApartamentosTuristicosNacional(@Path("
     periodo") String periodo);
@POST("area/{id}")
  Call<MapVieWideResponseCoordenadas> getAreaMapa(@Path("id")
     Integer id);
```

Las etiquetas de la parte superior de cada llamada indican el tipo de método HTTP y sobre qué ruta en el API se debe ejecutar. Como las peticiones deben ser asíncronas, el objeto que devuelve cada llamada es de tipo Call, el tipado del método al tipo de objeto que necesitemos.

A través de los parámetros que recibe el método le indicaremos los reemplazos que debe hacer en la URL mediante la etiqueta "@Path". Si la petición tiene cuerpo entonces el objeto deberá ser etiquetado como "@Body".

En el caso de ser necesarios los parámetros de la query, entonces se deberá indicar mediante la etiqueta "@QueryMap' y el objeto pasado sería de tipo "Map<String, String>".

Para hacer uso de estos métodos definidos en la interfaz, debemos de crear un objeto que implemente la interfaz a través del constructor de Retrofit, mediante el comando "new Retrofit.Builder()", a su vez le pasaremos la URL y el convertidor a Gson necesario para la representación en Json. A continuación vemos como utilizamos la interfaz anteriormente definida con los métodos del API:

Listado 6.11: Ejemplo de uso de la interfaz Retrofit.

```
Retrofit retrofit = new Retrofit.Builder().baseUrl(Constants.
   BASE_URL).addConverterFactory(
        GsonConverterFactory.create()).build();

  //Obtener datos del servidor (datos de la gráfica)
  private void executeDatosService(String idCCAA, String periodo,
     String metodoEjecutar,
      boolean compararNacional) {

```

```
8
9       DatosApiService apiDataService = new Builder().baseUrl(
            Constants.BASE_URL).addConverterFactory(
10           GsonConverterFactory.create()).build().create(
                DatosApiService.class);
11
12      Thread t = new Thread() {
13        public void run() {
14          Call<MapVieWideResponseDatos> mapVieWideResponseCall =
                null;
15          MapVieWideResponseDatos mapVieWideResponseDatos = null;
16          try {
17            mapVieWideResponseCall = invocarMetodo(idCCAA, periodo,
                  metodoEjecutar, apiDataService);
18          } catch (InvocationTargetException |
                IllegalAccessException | NoSuchMethodException e) {
19            e.printStackTrace();
20          }
21
22          try {
23            if (mapVieWideResponseCall != null) {
24              mapVieWideResponseDatos = mapVieWideResponseCall.
                    execute().body();
25              if(periodo != null && mapVieWideResponseDatos != null
                    && mapVieWideResponseDatos.getResultsDatos().get(0)
                    .getDatos().size() < Integer.parseInt(periodo)){
26                try {
27                  int nuevoPeriodo = Integer.parseInt(periodo) + 1;
28                  mapVieWideResponseCall = invocarMetodo(idCCAA,
                        String.valueOf(nuevoPeriodo), metodoEjecutar,
                        apiDataService);
29                  mapVieWideResponseDatos = mapVieWideResponseCall.
                        execute().body();
30                } catch (InvocationTargetException |
                      IllegalAccessException | NoSuchMethodException e)
                      {
31                  e.printStackTrace();
32                }
33              }
34              if (mapVieWideResponseDatos != null) {
35                if (!compararNacional) {
36                  listaDatosFinal = mapVieWideResponseDatos
37                      .getResultsDatos();
38                } else {
39                  listaDatosNacional = mapVieWideResponseDatos
40                      .getResultsDatos();
41                }
42              }
43            }
```

```
44            } catch (IOException e) {
45              e.printStackTrace();
46            }
47          }
48        };
49
50        t.start();
51        try {
52          t.join();
53        } catch (InterruptedException e) {
54          e.printStackTrace();
55        }
56      }
```

Podemos ver como se hace uso en la línea 9 de la interfaz creada mediante retrofit "DatosApiService". En la línea 24 se ejecuta la petición mediante el comando ".execute().body()"[5], lo que recuperará el cuerpo de la llamada a la API.

En la línea 36 se puede cómo se obtienen los datos después de que la ejecución haya sido correcta.

6.2.4. Clusters

Los clusters son una técnica utilizada para agrupar los marcadores en los mapas, lo que nos ayuda a administrar múltiples marcadores a diferentes niveles de zoom.

Cuando un usuario ve el mapa con un alto nivel de zoom, los marcadores individuales se muestran en el mapa.

Cuando el usuario se aleja del mapa, los marcadores se agrupan en grupos (cluster de marcadores) para facilitar la visualización del mapa y optimizar el rendimiento del mismo.

A continuación tenemos un ejemplo de los clusters en el mapa:

[5]Documentación Retrofit. `https://square.github.io/retrofit/`

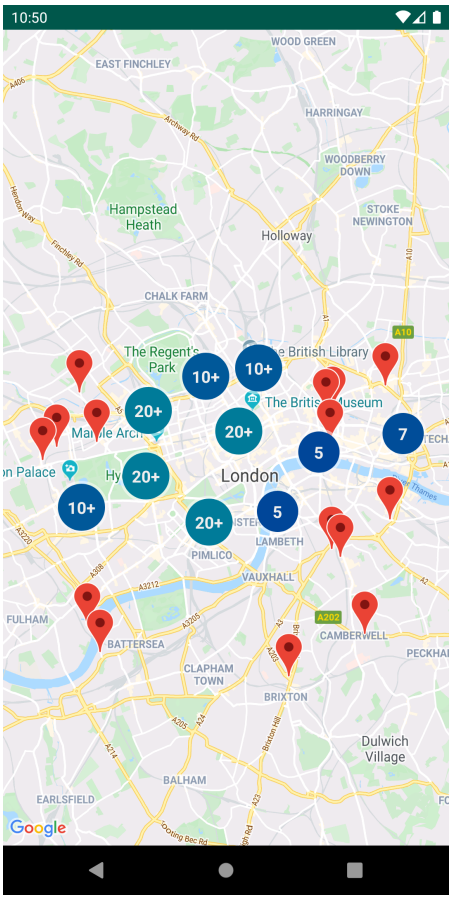


Figura 6.5.: Ejemplo de clusters en el mapa.

A continuación tenemos la implementación de la clase "MarkerClusterRenderer", la cual nos permite modificar el icono de los marcadores, al igual que añadirles un título y un snippet (descripción). También dentro de esta clase podemos indicar el tamaño que tendrá un cluster, normalmente se trabaja con múltiplos de 5, pero no siempre nos interesará hacerlo así.

También podemos indicarle cuantos marcadores deben reunirse en un cluster para que los englobe en un grupo.

Listado 6.12: Ejemplo de la clase MarkerClusterRenderer.

```
1 public class MarkerClusterRenderer extends DefaultClusterRenderer<
    ClusterMarker> {
2
3   public MarkerClusterRenderer(Context context, GoogleMap map,
      ClusterManager<ClusterMarker> clusterManager) {
```

```
4       super(context, map, clusterManager);
5     }
6
7     @Override
8     protected void onBeforeClusterItemRendered(ClusterMarker item,
          MarkerOptions markerOptions) {
9       markerOptions.icon(BitmapDescriptorFactory.fromResource(R.
            drawable.icon_ayuntamiento));
10      markerOptions.title(item.getTitle());
11      markerOptions.snippet(item.getSnippet());
12
13      super.onBeforeClusterItemRendered(item, markerOptions);
14    }
15
16    @Override
17    protected int getBucket(Cluster<ClusterMarker> cluster) {
18      return cluster.getSize();
19    }
20
21    @Override
22    protected boolean shouldRenderAsCluster(Cluster<ClusterMarker>
          cluster) {
23      return cluster.getSize() >= 3;
24    }
25
26    @Override
27    protected String getClusterText(int bucket) {
28      return String.valueOf(bucket);
29    }
30  }
```

A la hora de querer añadir los marcadores al "ClusterManager" para que los pueda agrupar, los añadiremos como si fuese un arrayList, pero no debemos de olvidarnos de realizar la llamada al método "clusterManager.cluster()" para que se actualicen todos los items que pertenecen al grupo.

Aquí podemos ver un ejemplo:

Listado 6.13: Ejemplo de añadir nuevos marcadores al Cluster Manager.

```
1  ClusterMarker newClusterMarker = new ClusterMarker(
2              new LatLng(address.getLatitude(), address.getLongitude
                   ()), address.getFeatureName(), "");
3  clusterManager.addItem(newClusterMarker);
4  clusterManager.cluster();
```

7. Conclusiones

Tras haber explicado cada apartado del proyecto, voy a proceder a repasar el trabajo realizado en el mismo, lo que ha supuesto para mí y para lo que me ha servido.

Desde el punto de vista de desarrollador he de decir que he aprendido mucho, ha sido una experiencia muy enriquecedora, sobre todo por ser una tecnología que nunca antes había tocado como es el caso de Android y por ser un proyecto que he realizado yo solo, sin compañeros ni ayuda externa, salvo preguntas puntuales al tutor, pero desde el punto de vista del desarrollo. También he aprendido mucho de mis errores y de como solventarlos de cara al futuro.

En el primer apartado (ver sección 1.1) hablamos sobre los objetivos que debería cumplir la aplicación cuando estuviese acabada, a continuación se analiza qué es lo que hemos podido conseguir de lo que nos habíamos propuesto inicialmente:

En primer lugar tenemos los objetivos del desarrollo software:

- Desarrollar un código limpio, modular y reutilizable: se ha utilizado el patrón MVVM que nos ha ayudado a la organización del código en secciones, también por mi cuenta, he ido refactorizando el código conforme lo he ido escribiendo, por lo que hemos mantenido una coherencia dentro del código evitando el conocido "Código espagueti".

- Mantener código documentado con comentarios: se han ido añadiendo comentarios para el código más complejo, además de para las librerías externas que se han utilizado para que de cara al futuro sea reutilizable el código.

- Realizar todas las pruebas que sean posibles: en este caso, se han realizado pruebas conforme se ha ido desarrollando el código, no se han creado pruebas unitarias ni de integración, pero sí se ha ido probando cada funcionalidad tras ser añadida.

- Hacer uso de las últimas versiones de librerías y frameworks: Se han utilizado todas las librerías actualizadas en su última versión desde el inicio del proyecto.

- Gestionar el desarrollo mediante Git: en todo el desarrollo se ha utilizado Git para

controlar las distintas versiones del código, por seguridad y también para tener mejor organizado el desarrollo.

En segundo lugar tenemos los objetivos del desarrollo de la aplicación:

- Desarrollar una aplicación que muestre las fuentes de información disponibles: Mediante la gráfica mostramos todas las fuentes de información disponibles del API.
- Permitir al usuario mostrar los datos que le interesan: Mediante los elementos del menú somos capaces de filtrar la información deseada.
- Comparar los datos mostrados: Somos capaces de comparar los resultados por comunidad con los nacionales.

En tercer lugar tenemos los objetivos de la metodología de desarrollo:

- Reunir todos los requisitos y tareas necesarias para el desarrollo: esto lo hemos conseguido gracias a las herramientas mostradas en el apartado 4, en concreto con Balsamiq y Draw.io.
- Utilizar control de versiones: Hemos utilizado Git como controlador de versiones así como sus herramientas disponibles (Issues, hitos...).
- Ser capaz de ver el estado de la app de un vistazo: Mediante los tableros de GitHub esto es posible como se puede observar en la imagen 4.12.
- Versionar la aplicación y lanzar varias releases: Esto lo hemos conseguido lanzando versiones tanto de nuevas features como arreglando bugs que han ido apareciendo durante el desarrollo, que finalmente constituyen una nueva versión o lanzamiento de una release.

A pesar de todos los objetivos que se han cumplido, se podría mejorar mucho más la aplicación de cara al futuro en diversos aspectos, como recomendaciones, se podrían realizar las siguientes mejoras:

- Realizar pruebas unitarias, de integración y de regresión en el proyecto para tener un mínimo de consistencia en los cambios que se fueran a realizar en el futuro.
- Añadir una base de datos que contenga los datos que actualmente se almacenan en caché para que sea más rápido el acceso a los mismos. Añadir seguridad en este caso.

- Crear un login para que los usuarios puedan acceder a sus cuentas y dentro poder guardarse datos o comunidades como favoritos u otras características.

- Ser una aplicación retroactiva para que todos los usuarios puedan aportar información a la misma, habría que verificar que la información sea verídica.

A. Imágenes de la aplicación final

A continuación se mostrarán algunas imágenes de la aplicación en funcionamiento:

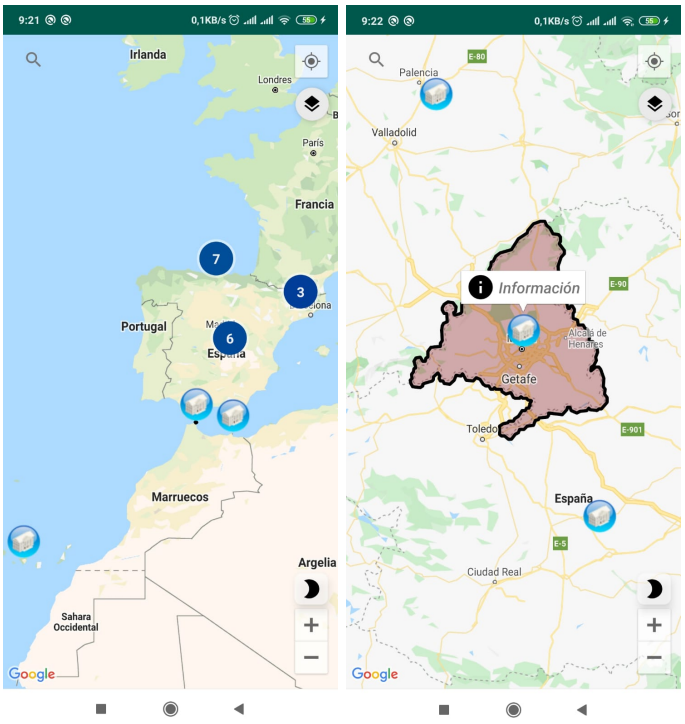


Figura A.1.: Vista general del mapa y área de una comunidad autónoma.

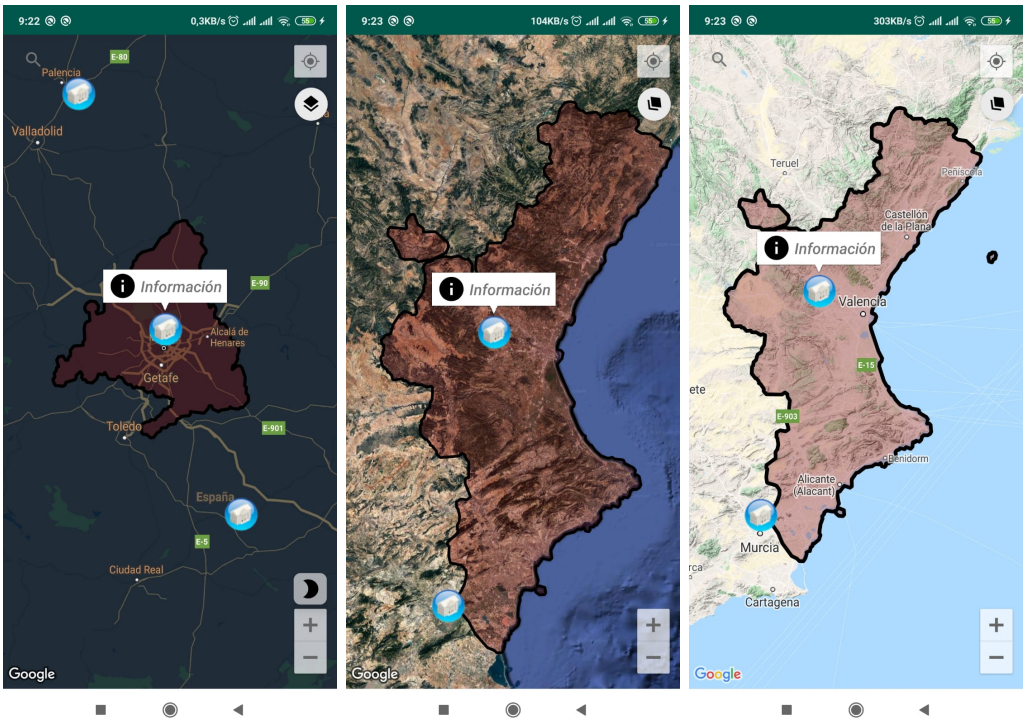


Figura A.2.: Mostrando modo noche de la aplicación y terrenos de la aplicación.

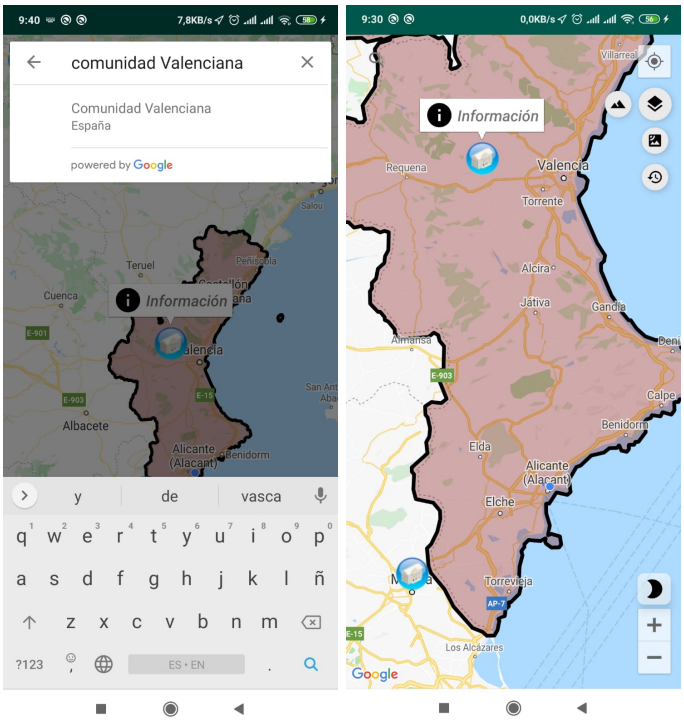


Figura A.3.: Buscador, geolocalización y botones para cambiar el terreno.

Figura A.4.: Mostrando datos de la aplicación comparando y sin hacerlo.

B. Documentación de la API

A continuación se mostrará la documentación completa de la API en el lenguaje YAML creado por Swagger:

```yaml
swagger: "2.0"
info:
  description: "Documentación sobre el API de la aplicación
     MapVieWide"
  version: "1.0.0"
  title: "MapVieWide"
  contact:
    email: "netrik@hotmail.es"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
host: "localhost:8000"
basePath: "/v1"
tags:
- name: "Datos mapa"
  description: "Datos sobre CCAA de la aplicación obtenidos del
     INE (Instituto nacional de estadística)"
- name: "Datos mapa Nacional"
  description: "Datos nacionales de la aplicación obtenidos del
     INE (Instituto nacional de estadística)"
- name: "Área mapa"
  description: "Datos sobre áreas del mapa (polígonos) obtenidos
     de OpenStreetMap"
paths:
  /area/{id}:
    post:
      tags:
      - "Área mapa"
      summary: "Obtenemos el respectivo área en puntos"
      operationId: "area/id"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
```

```
33        parameters:
34        - name: "id"
35          in: "path"
36          description: "ID de la CCAA deseada"
37          required: true
38          type: "string"
39        responses:
40          "200":
41            description: "Operación realizada correctamente"
42          "404":
43            description: "Dirección no encontrada, revisar URL"
44          "500":
45            description: "Error interno en el servidor, revisar URL"
46
47    /poblacionCCAA/{idCCAA}:
48      get:
49        tags:
50        - "Datos mapa"
51        summary: "Obtenemos el dato de población sobre un CCAA espec
              ífico"
52        operationId: "getPoblacionCCAA/idCCAA"
53        consumes:
54        - "application/json"
55        - "application/xml"
56        produces:
57        - "application/xml"
58        - "application/json"
59        parameters:
60        - name: "idCCAA"
61          in: "path"
62          description: "Filtro por ID de la comunidad autónoma"
63          required: true
64          type: "string"
65        responses:
66          "200":
67            description: "Operación realizada correctamente"
68          "404":
69            description: "Dirección no encontrada, revisar URL"
70          "500":
71            description: "Error interno en el servidor, revisar URL"
72
73
74    /poblacionCCAA/{idCCAA}/{periodo}:
75      get:
76        tags:
77        - "Datos mapa"
78        summary: "Obtenemos el dato de población sobre un CCAA y per
              íodo específico"
79        operationId: "getPoblacionCCAA/idCCAA/periodo"
```

```
80        consumes:
81        - "application/json"
82        - "application/xml"
83        produces:
84        - "application/xml"
85        - "application/json"
86        parameters:
87        - name: "idCCAA"
88          in: "path"
89          description: "Filtro por ID de la comunidad autónoma"
90          required: true
91          type: "string"
92        - name: "periodo"
93          in: "path"
94          description: "Periodo de la petición, años a mostrar"
95          required: false
96          type: "string"
97        responses:
98          "200":
99            description: "Operación realizada correctamente"
100         "404":
101           description: "Dirección no encontrada, revisar URL"
102         "500":
103           description: "Error interno en el servidor, revisar URL"
104
105   /natalidadCCAA/{idCCAA}:
106     get:
107       tags:
108       - "Datos mapa"
109       summary: "Obtenemos el dato de natalidad sobre un CCAA espec
              ífico"
110       operationId: "getNatalidadCCAA/idCCAA"
111       consumes:
112       - "application/json"
113       - "application/xml"
114       produces:
115       - "application/xml"
116       - "application/json"
117       parameters:
118       - name: "idCCAA"
119         in: "path"
120         description: "Filtro por ID de la comunidad autónoma"
121         required: true
122         type: "string"
123       responses:
124         "200":
125           description: "Operación realizada correctamente"
126         "404":
127           description: "Dirección no encontrada, revisar URL"
```

```
128           "500":
129             description: "Error interno en el servidor, revisar URL"
130
131
132     /natalidadCCAA/{idCCAA}/{periodo}:
133       get:
134         tags:
135         - "Datos mapa"
136         summary: "Obtenemos el dato de natalidad sobre un CCAA y per
                íodo específico"
137         operationId: "getNatalidadCCAA/idCCAA/periodo"
138         consumes:
139         - "application/json"
140         - "application/xml"
141         produces:
142         - "application/xml"
143         - "application/json"
144         parameters:
145         - name: "idCCAA"
146           in: "path"
147           description: "Filtro por ID de la comunidad autónoma"
148           required: true
149           type: "string"
150         - name: "periodo"
151           in: "path"
152           description: "Periodo de la petición, años a mostrar"
153           required: false
154           type: "string"
155         responses:
156           "200":
157             description: "Operación realizada correctamente"
158           "404":
159             description: "Dirección no encontrada, revisar URL"
160           "500":
161             description: "Error interno en el servidor, revisar URL"
162
163     /mortalidadCCAA/{idCCAA}:
164       get:
165         tags:
166         - "Datos mapa"
167         summary: "Obtenemos el dato de natalidad sobre un CCAA espec
                ífico"
168         operationId: "getMortalidadCCAA/idCCAA"
169         consumes:
170         - "application/json"
171         - "application/xml"
172         produces:
173         - "application/xml"
174         - "application/json"
```

```
175         parameters:
176         - name: "idCCAA"
177           in: "path"
178           description: "Filtro por ID de la comunidad autónoma"
179           required: true
180           type: "string"
181         responses:
182           "200":
183             description: "Operación realizada correctamente"
184           "404":
185             description: "Dirección no encontrada, revisar URL"
186           "500":
187             description: "Error interno en el servidor, revisar URL"
188 
189 
190     /mortalidadCCAA/{idCCAA}/{periodo}:
191       get:
192         tags:
193         - "Datos mapa"
194         summary: "Obtenemos el dato de natalidad sobre un CCAA y per
                íodo específico"
195         operationId: "getMortalidadCCAA/idCCAA/periodo"
196         consumes:
197         - "application/json"
198         - "application/xml"
199         produces:
200         - "application/xml"
201         - "application/json"
202         parameters:
203         - name: "idCCAA"
204           in: "path"
205           description: "Filtro por ID de la comunidad autónoma"
206           required: true
207           type: "string"
208         - name: "periodo"
209           in: "path"
210           description: "Periodo de la petición, años a mostrar"
211           required: false
212           type: "string"
213         responses:
214           "200":
215             description: "Operación realizada correctamente"
216           "404":
217             description: "Dirección no encontrada, revisar URL"
218           "500":
219             description: "Error interno en el servidor, revisar URL"
220 
221     /densidadCCAA/{idCCAA}:
222       get:
```

```
223         tags:
224         - "Datos mapa"
225         summary: "Obtenemos el dato de densidad sobre un CCAA específico"
226         operationId: "getDensidadCCAA/idCCAA"
227         consumes:
228         - "application/json"
229         - "application/xml"
230         produces:
231         - "application/xml"
232         - "application/json"
233         parameters:
234         - name: "idCCAA"
235           in: "path"
236           description: "Filtro por ID de la comunidad autónoma"
237           required: true
238           type: "string"
239         responses:
240           "200":
241             description: "Operación realizada correctamente"
242           "404":
243             description: "Dirección no encontrada, revisar URL"
244           "500":
245             description: "Error interno en el servidor, revisar URL"
246
247
248     /densidadCCAA/{idCCAA}/{periodo}:
249       get:
250         tags:
251         - "Datos mapa"
252         summary: "Obtenemos el dato de densidad sobre un CCAA y período específico"
253         operationId: "getDensidadCCAA/idCCAA/periodo"
254         consumes:
255         - "application/json"
256         - "application/xml"
257         produces:
258         - "application/xml"
259         - "application/json"
260         parameters:
261         - name: "idCCAA"
262           in: "path"
263           description: "Filtro por ID de la comunidad autónoma"
264           required: true
265           type: "string"
266         - name: "periodo"
267           in: "path"
268           description: "Periodo de la petición, años a mostrar"
269           required: false
```

```
270           type: "string"
271         responses:
272           "200":
273             description: "Operación realizada correctamente"
274           "404":
275             description: "Dirección no encontrada, revisar URL"
276           "500":
277             description: "Error interno en el servidor, revisar URL"
278
279     /envejecimientoCCAA/{idCCAA}:
280       get:
281         tags:
282         - "Datos mapa"
283         summary: "Obtenemos el dato de envejecimiento sobre un CCAA
                específico"
284         operationId: "getEnvejecimientoCCAA/idCCAA"
285         consumes:
286         - "application/json"
287         - "application/xml"
288         produces:
289         - "application/xml"
290         - "application/json"
291         parameters:
292         - name: "idCCAA"
293           in: "path"
294           description: "Filtro por ID de la comunidad autónoma"
295           required: true
296           type: "string"
297         responses:
298           "200":
299             description: "Operación realizada correctamente"
300           "404":
301             description: "Dirección no encontrada, revisar URL"
302           "500":
303             description: "Error interno en el servidor, revisar URL"
304
305
306     /envejecimientoCCAA/{idCCAA}/{periodo}:
307       get:
308         tags:
309         - "Datos mapa"
310         summary: "Obtenemos el dato de envejecimiento sobre un CCAA
                y período específico"
311         operationId: "getEnvejecimientoCCAA/idCCAA/periodo"
312         consumes:
313         - "application/json"
314         - "application/xml"
315         produces:
316         - "application/xml"
```

```
317         - "application/json"
318         parameters:
319         - name: "idCCAA"
320           in: "path"
321           description: "Filtro por ID de la comunidad autónoma"
322           required: true
323           type: "string"
324         - name: "periodo"
325           in: "path"
326           description: "Periodo de la petición, años a mostrar"
327           required: false
328           type: "string"
329         responses:
330           "200":
331             description: "Operación realizada correctamente"
332           "404":
333             description: "Dirección no encontrada, revisar URL"
334           "500":
335             description: "Error interno en el servidor, revisar URL"
336
337     /migracionInteriorCCAA/{idCCAA}:
338       get:
339         tags:
340         - "Datos mapa"
341         summary: "Obtenemos el dato de migracion interior sobre un
                CCAA específico"
342         operationId: "getMigracionInteriorCCAA/idCCAA"
343         consumes:
344         - "application/json"
345         - "application/xml"
346         produces:
347         - "application/xml"
348         - "application/json"
349         parameters:
350         - name: "idCCAA"
351           in: "path"
352           description: "Filtro por ID de la comunidad autónoma"
353           required: true
354           type: "string"
355         responses:
356           "200":
357             description: "Operación realizada correctamente"
358           "404":
359             description: "Dirección no encontrada, revisar URL"
360           "500":
361             description: "Error interno en el servidor, revisar URL"
362
363
364     /migracionInteriorCCAA/{idCCAA}/{periodo}:
```

```
365       get:
366         tags:
367         - "Datos mapa"
368         summary: "Obtenemos el dato de migracion interior sobre un
               CCAA y período específico"
369         operationId: "getMigracionInteriorCCAA/idCCAA/periodo"
370         consumes:
371         - "application/json"
372         - "application/xml"
373         produces:
374         - "application/xml"
375         - "application/json"
376         parameters:
377         - name: "idCCAA"
378           in: "path"
379           description: "Filtro por ID de la comunidad autónoma"
380           required: true
381           type: "string"
382         - name: "periodo"
383           in: "path"
384           description: "Periodo de la petición, años a mostrar"
385           required: false
386           type: "string"
387         responses:
388           "200":
389             description: "Operación realizada correctamente"
390           "404":
391             description: "Dirección no encontrada, revisar URL"
392           "500":
393             description: "Error interno en el servidor, revisar URL"
394
395     /migracionExteriorCCAA/{idCCAA}:
396       get:
397         tags:
398         - "Datos mapa"
399         summary: "Obtenemos el dato de migracion exterior sobre un
               CCAA específico"
400         operationId: "getMigracionExteriorCCAA/idCCAA"
401         consumes:
402         - "application/json"
403         - "application/xml"
404         produces:
405         - "application/xml"
406         - "application/json"
407         parameters:
408         - name: "idCCAA"
409           in: "path"
410           description: "Filtro por ID de la comunidad autónoma"
411           required: true
```

```
412           type: "string"
413         responses:
414           "200":
415             description: "Operación realizada correctamente"
416           "404":
417             description: "Dirección no encontrada, revisar URL"
418           "500":
419             description: "Error interno en el servidor, revisar URL"
420
421
422     /migracionExteriorCCAA/{idCCAA}/{periodo}:
423       get:
424         tags:
425         - "Datos mapa"
426         summary: "Obtenemos el dato de migracion exterior sobre un
                CCAA y período específico"
427         operationId: "getMigracionExteriorCCAA/idCCAA/periodo"
428         consumes:
429         - "application/json"
430         - "application/xml"
431         produces:
432         - "application/xml"
433         - "application/json"
434         parameters:
435         - name: "idCCAA"
436           in: "path"
437           description: "Filtro por ID de la comunidad autónoma"
438           required: true
439           type: "string"
440         - name: "periodo"
441           in: "path"
442           description: "Periodo de la petición, años a mostrar"
443           required: false
444           type: "string"
445         responses:
446           "200":
447             description: "Operación realizada correctamente"
448           "404":
449             description: "Dirección no encontrada, revisar URL"
450           "500":
451             description: "Error interno en el servidor, revisar URL"
452
453     /preciosTurismoRuralCCAA/{idCCAA}:
454       get:
455         tags:
456         - "Datos mapa"
457         summary: "Obtenemos el dato de precios turismo rural sobre
                un CCAA específico"
458         operationId: "getPreciosTurismoRealCCAA/idCCAA"
```

```
459         consumes:
460         - "application/json"
461         - "application/xml"
462         produces:
463         - "application/xml"
464         - "application/json"
465         parameters:
466         - name: "idCCAA"
467           in: "path"
468           description: "Filtro por ID de la comunidad autónoma"
469           required: true
470           type: "string"
471         responses:
472           "200":
473             description: "Operación realizada correctamente"
474           "404":
475             description: "Dirección no encontrada, revisar URL"
476           "500":
477             description: "Error interno en el servidor, revisar URL"
478
479
480     /preciosTurismoRuralCCAA/{idCCAA}/{periodo}:
481       get:
482         tags:
483         - "Datos mapa"
484         summary: "Obtenemos el dato de precios turismo rural sobre
                un CCAA y período específico"
485         operationId: "getPreciosTurismoRealCCAA/idCCAA/periodo"
486         consumes:
487         - "application/json"
488         - "application/xml"
489         produces:
490         - "application/xml"
491         - "application/json"
492         parameters:
493         - name: "idCCAA"
494           in: "path"
495           description: "Filtro por ID de la comunidad autónoma"
496           required: true
497           type: "string"
498         - name: "periodo"
499           in: "path"
500           description: "Periodo de la petición, años a mostrar"
501           required: false
502           type: "string"
503         responses:
504           "200":
505             description: "Operación realizada correctamente"
506           "404":
```

```
507             description: "Dirección no encontrada, revisar URL"
508           "500":
509             description: "Error interno en el servidor, revisar URL"
510
511     /preciosHotelesCCAA/{idCCAA}:
512       get:
513         tags:
514         - "Datos mapa"
515         summary: "Obtenemos el dato de precios hoteles sobre un CCAA
                específico"
516         operationId: "getPreciosHotelesCCAA/idCCAA"
517         consumes:
518         - "application/json"
519         - "application/xml"
520         produces:
521         - "application/xml"
522         - "application/json"
523         parameters:
524         - name: "idCCAA"
525           in: "path"
526           description: "Filtro por ID de la comunidad autónoma"
527           required: true
528           type: "string"
529         responses:
530           "200":
531             description: "Operación realizada correctamente"
532           "404":
533             description: "Dirección no encontrada, revisar URL"
534           "500":
535             description: "Error interno en el servidor, revisar URL"
536
537
538     /preciosHotelesCCAA/{idCCAA}/{periodo}:
539       get:
540         tags:
541         - "Datos mapa"
542         summary: "Obtenemos el dato de precios hoteles sobre un CCAA
                y período específico"
543         operationId: "getPreciosHotelesCCAA/idCCAA/periodo"
544         consumes:
545         - "application/json"
546         - "application/xml"
547         produces:
548         - "application/xml"
549         - "application/json"
550         parameters:
551         - name: "idCCAA"
552           in: "path"
553           description: "Filtro por ID de la comunidad autónoma"
```

```
554           required: true
555           type: "string"
556         - name: "periodo"
557           in: "path"
558           description: "Periodo de la petición, años a mostrar"
559           required: false
560           type: "string"
561         responses:
562           "200":
563             description: "Operación realizada correctamente"
564           "404":
565             description: "Dirección no encontrada, revisar URL"
566           "500":
567             description: "Error interno en el servidor, revisar URL"
568
569     /preciosTarifasApartamentosTuristicosNacional/{periodo}:
570       get:
571         tags:
572         - "Datos mapa Nacional"
573         summary: "Obtenemos el dato de precios de tarifas en
                apartamentos turisticos a nivel nacional de un período
                específico"
574         operationId: "
                getPreciosTarifasApartamentosTuristicosNacional"
575         consumes:
576         - "application/json"
577         - "application/xml"
578         produces:
579         - "application/xml"
580         - "application/json"
581         parameters:
582         - name: "periodo"
583           in: "path"
584           description: "Periodo de la petición, años a mostrar"
585           required: true
586           type: "string"
587         responses:
588           "200":
589             description: "Operación realizada correctamente"
590           "404":
591             description: "Dirección no encontrada, revisar URL"
592           "500":
593             description: "Error interno en el servidor, revisar URL"
594
595     /preciosTarifasCampingsNacional/{periodo}:
596       get:
597         tags:
598         - "Datos mapa Nacional"
599         summary: "Obtenemos el dato de precios de tarifas en
```

```
          campings a nivel nacional de un período específico"
      operationId: "
          getPreciosTarifasApartamentosTuristicosNacional"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - name: "periodo"
        in: "path"
        description: "Periodo de la petición, años a mostrar"
        required: true
        type: "string"
      responses:
        "200":
          description: "Operación realizada correctamente"
        "404":
          description: "Dirección no encontrada, revisar URL"
        "500":
          description: "Error interno en el servidor, revisar URL"

  /gastoMedioNacional/{periodo}:
    get:
      tags:
      - "Datos mapa Nacional"
      summary: "Obtenemos el dato de gasto medio a nivel nacional
          de un período específico"
      operationId: "getGastoMedioNacional"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - name: "periodo"
        in: "path"
        description: "Periodo de la petición, años a mostrar"
        required: true
        type: "string"
      responses:
        "200":
          description: "Operación realizada correctamente"
        "404":
          description: "Dirección no encontrada, revisar URL"
        "500":
          description: "Error interno en el servidor, revisar URL"
```

```
646
647   /rentaMediaFamiliaCCAA/{idCCAA}:
648     get:
649       tags:
650       - "Datos mapa"
651       summary: "Obtenemos el dato de renta media familiar sobre un
              CCAA específico"
652       operationId: "getRentaMediaFamiliaCCAA/idCCAA"
653       consumes:
654       - "application/json"
655       - "application/xml"
656       produces:
657       - "application/xml"
658       - "application/json"
659       parameters:
660       - name: "idCCAA"
661         in: "path"
662         description: "Filtro por ID de la comunidad autónoma"
663         required: true
664         type: "string"
665       responses:
666         "200":
667           description: "Operación realizada correctamente"
668         "404":
669           description: "Dirección no encontrada, revisar URL"
670         "500":
671           description: "Error interno en el servidor, revisar URL"
672
673
674   /rentaMediaFamiliaCCAA/{idCCAA}/{periodo}:
675     get:
676       tags:
677       - "Datos mapa"
678       summary: "Obtenemos el dato de renta media familiar sobre un
              CCAA y período específico"
679       operationId: "getRentaMediaFamiliaCCAA/idCCAA/periodo"
680       consumes:
681       - "application/json"
682       - "application/xml"
683       produces:
684       - "application/xml"
685       - "application/json"
686       parameters:
687       - name: "idCCAA"
688         in: "path"
689         description: "Filtro por ID de la comunidad autónoma"
690         required: true
691         type: "string"
692       - name: "periodo"
```

```
693           in: "path"
694           description: "Periodo de la petición, años a mostrar"
695           required: false
696           type: "string"
697         responses:
698           "200":
699             description: "Operación realizada correctamente"
700           "404":
701             description: "Dirección no encontrada, revisar URL"
702           "500":
703             description: "Error interno en el servidor, revisar URL"
704
705     /rentaMediaPersonaCCAA/{idCCAA}:
706       get:
707         tags:
708         - "Datos mapa"
709         summary: "Obtenemos el dato de renta media por persona sobre
                un CCAA específico"
710         operationId: "getRentaMediaPersonaCCAA/idCCAA"
711         consumes:
712         - "application/json"
713         - "application/xml"
714         produces:
715         - "application/xml"
716         - "application/json"
717         parameters:
718         - name: "idCCAA"
719           in: "path"
720           description: "Filtro por ID de la comunidad autónoma"
721           required: true
722           type: "string"
723         responses:
724           "200":
725             description: "Operación realizada correctamente"
726           "404":
727             description: "Dirección no encontrada, revisar URL"
728           "500":
729             description: "Error interno en el servidor, revisar URL"
730
731
732     /rentaMediaPersonaCCAA/{idCCAA}/{periodo}:
733       get:
734         tags:
735         - "Datos mapa"
736         summary: "Obtenemos el dato de renta media por persona sobre
                un CCAA y período específico"
737         operationId: "getRentaMediaPersonaCCAA/idCCAA/periodo"
738         consumes:
739         - "application/json"
```

```
740         - "application/xml"
741         produces:
742         - "application/xml"
743         - "application/json"
744         parameters:
745         - name: "idCCAA"
746           in: "path"
747           description: "Filtro por ID de la comunidad autónoma"
748           required: true
749           type: "string"
750         - name: "periodo"
751           in: "path"
752           description: "Periodo de la petición, años a mostrar"
753           required: false
754           type: "string"
755         responses:
756           "200":
757             description: "Operación realizada correctamente"
758           "404":
759             description: "Dirección no encontrada, revisar URL"
760           "500":
761             description: "Error interno en el servidor, revisar URL"
762
763     /riesgoPobrezaCCAA/{idCCAA}:
764       get:
765         tags:
766         - "Datos mapa"
767         summary: "Obtenemos el dato de riesgo de pobreza sobre un
                CCAA específico"
768         operationId: "getRiesgoPobrezaCCAA/idCCAA"
769         consumes:
770         - "application/json"
771         - "application/xml"
772         produces:
773         - "application/xml"
774         - "application/json"
775         parameters:
776         - name: "idCCAA"
777           in: "path"
778           description: "Filtro por ID de la comunidad autónoma"
779           required: true
780           type: "string"
781         responses:
782           "200":
783             description: "Operación realizada correctamente"
784           "404":
785             description: "Dirección no encontrada, revisar URL"
786           "500":
787             description: "Error interno en el servidor, revisar URL"
```

```
788
789
790   /riesgoPobrezaCCAA/{idCCAA}/{periodo}:
791     get:
792       tags:
793       - "Datos mapa"
794       summary: "Obtenemos el dato de riesgo de pobreza sobre un
              CCAA y período específico"
795       operationId: "getRiesgoPobrezaCCAA/idCCAA/periodo"
796       consumes:
797       - "application/json"
798       - "application/xml"
799       produces:
800       - "application/xml"
801       - "application/json"
802       parameters:
803       - name: "idCCAA"
804         in: "path"
805         description: "Filtro por ID de la comunidad autónoma"
806         required: true
807         type: "string"
808       - name: "periodo"
809         in: "path"
810         description: "Periodo de la petición, años a mostrar"
811         required: false
812         type: "string"
813       responses:
814         "200":
815           description: "Operación realizada correctamente"
816         "404":
817           description: "Dirección no encontrada, revisar URL"
818         "500":
819           description: "Error interno en el servidor, revisar URL"
```